Draper Pascal

Version 2.1

For the Atari 400, 800, XL, and XE series computers

Table of Contents

Note: An asterisk (*) following the page number indicates that the item was either not present, or only briefly explained, in the "Starter" version of the Draper Pascal manual.

Introduction ---------- 4
  What is Pascal? ---------- 4
  What is Draper Pascal? ---------- 4
  About this manual ---------- 4
  What is Draper Pascal made of? ---------- 4
  About the DOS ---------- 4
  The ShareWare Concept ---------- 5
Ramdisk support ---------- 6 *
Getting Started ---------- 8
The Main Menu ---------- 13
  1 - Run Program ---------- 13
  2 - Disk Directory ---------- 13
  3 - Compile Program ---------- 14
  4 - Edit a Program ---------- 14
  5 - Exit to DOS ---------- 14
  6 - List a file ---------- 14
  7 - Trace on ---------- 14
The Editor ---------- 15
  General Prompts ---------- 15
  The Commands ---------- 16
    A - Add line(s) at end ---------- 16
    C - Change line(s) ---------- 16
    D - Delete line(s) ---------- 16
    E - Edit line(s) ---------- 17
    F - Filer menu ---------- 17
      A - Append file ---------- 17
      D - Directory list ---------- 17
      L - Load file ---------- 17
      S - Save file ---------- 17
    I - Insert before line ---------- 17
    L - List line(s) ---------- 18
    M - Menu ---------- 18
    P - Print line(s) ---------- 18
    Q - Quit ---------- 18
    S - Scan line(s) ---------- 18
    X - Exit to Compiler ---------- 18
The Compiler ---------- 19
The Supervisor ---------- 22
Pascal Definitions ---------- 23
  ABS ---------- 23 *
  ADDR ---------- 23 *
  AND ---------- 23 *
  ARCTAN ---------- 23 *
  ARRAY ---------- 24 *
  ASC ---------- 24 *
  BEGIN ---------- 25 *
  BLOAD ---------- 25 *
  BOOLEAN ---------- 26 *
  CALL ---------- 27 *
  CASE ---------- 27 *
  CHAR ---------- 28 *
  CHR ---------- 28 *
  CLOSE ---------- 28 *
  COLOR ---------- 29 *
  CONCAT ---------- 29 *
  CONST ---------- 29 *
  COPY ---------- 29 *
  COS ---------- 30 *
  CVTREAL ---------- 30 *
  DEG ---------- 30 *
  DELETE ---------- 31 *
  DIV ---------- 31 *
  DOS ---------- 32 *
  DRAWTO ---------- 32 *

DUMPSTK ---------- 32 *
DVSTAT ---------- 33 *
END ---------- 33 *
EOF ---------- 33 *
EOLN ---------- 34 *
EXIT ---------- 34 *
EXP ---------- 34 *
EXP10 ---------- 34 *
FALSE ---------- 35 *
FILE ---------- 35 *
FOR ---------- 35 *
FUNCTION ---------- 36 *
GOTOXY ---------- 36 *
GRAPHICS ---------- 36 *
HIMEM ---------- 37 *
IF ---------- 37 *
INSERT ---------- 37 *
INTEGER ---------- 38 *
IORESULT ---------- 38 *
KEYPRESS ---------- 38 *
LENGTH ---------- 38 *
LN ---------- 39 *
LOCATE ---------- 39 *
LOCK ---------- 39 *
LOG ---------- 40 *
LPENH ---------- 40 *
LPENV ---------- 40 *
MAXGRAPH ---------- 40 *
MOD ---------- 41 *
NOT ---------- 41 *
NOTE ---------- 41 *
ODD ---------- 42 *
OPEN ---------- 42 *
OPTIONKEY ---------- 43 *
OPTIONS ---------- 43 *
OR ---------- 44 *
ORD ---------- 44 *
PADDLE ---------- 45 *
PEEK ---------- 45 *
PLOT ---------- 45 *
POINT ---------- 45 *
POKE ---------- 46 *
POS ---------- 46 *
PROCEDURE ---------- 46 *
PROGRAM ---------- 47 *
PTRIG ---------- 47 *
PURGE ---------- 47 *
RAD ---------- 48 *
READ ---------- 48 *
READLN ---------- 48 *
REAL ---------- 48 *
RECORD ---------- 49 *
REPEAT ---------- 49 *
RESET ---------- 50 *
REWRITE ---------- 50 *
RND ---------- 50 *
SELECTKEY ---------- 50 *
SETCOLOR ---------- 51 *
SHL ---------- 51 *
SHR ---------- 51 *
SIN ---------- 52 *
SOUND ---------- 52 *
SQR ---------- 52 *
SQRT ---------- 53 *
STARTKEY ---------- 53 *
STATUS ---------- 53 *
STICK ---------- 53 *
STR ---------- 54 *
STRIG ---------- 54 *
STRING ---------- 54 *

TRACEOFF -- 55 *
TRACEON --- 55 *
TRUE -- 55 *
UNLOCK -- 56 *
VAL --- 56 *
VAR --- 56 *
WAIT -- 57 *
WHILE --- 57 *
WRITE --- 57 *
WRITELN --- 58 *
XCTL -- 59 *
XIO --- 59 *
System Information ------------------------------------ 61 *
Internal Data Formats --------------------------------- 61 *
Suppressing the Title Screen -------------------------- 62 *
Trace Format -- 62 *
Reserved Word List ------------------------------------ 63 *
Operators --- 64
Editor Command Summary -------------------------------- 65 *
Error Messages -- 66
Compile Time Error Messages --------------------------- 66
Execution Time Error Messages ------------------------- 67
Main Menu Program Source Listing ---------------------- 69 *
Editor Program Source Listing ------------------------- 72 *
Ramdisk Program Source Listings ----------------------- 79 *
Sample Program Source Listings ------------------------ 81 *
Printer Usage --- 83 *
Software License -------------------------------------- 84 *

Introduction

Draper Software welcomes you to the world of Pascal for the Atari 400/800, XL, and XE series Computer systems.

What is Pascal?

Pascal is a high-level structured programming language developed by Niklaus Wirth in 1971. It is easy to understand and well suited for program development and maintenance.

What is Draper Pascal?

Draper Pascal is not a "standard" Pascal. It has a number of commands which are exactly like ISO and UCSD versions, some which are similar, and many "extensions" which bring out the true power of the Atari computer in an easy to use manner. It was designed to require only one disk drive for operation, but not be limited to only one. At this time, it has been shown to work with all hardware and software configurations where enough memory is provided. This implementation also has a number of commands which are familiar to Atari BASIC users, such as POKE, PEEK, SETCOLOR, NOTE, POINT, etc..

About this manual

This manual is intended to familiarize you with all the features of Draper Pascal. It is not intended to teach you how to program in Pascal. However, if you already know Atari BASIC, then you can understand the Pascal statements more easily by referring to their BASIC equivalents shown after the definition of each Pascal reserved word. It is recommended that you read this manual completely to be familiarized with its features and restrictions.

What is Draper Pascal made of?

This implementation of Pascal is made up of three main components. They are the Supervisor (sometimes referred to as runtime routines), the Compiler, and the Editor. The Supervisor is a high performance machine language program which simulates a 16-bit pseudo computer. The Compiler translates Pascal source code into pseudo-code instructions to be executed by the Supervisor. The Editor is used to enter and modify Pascal source programs. It may also be used to edit data files, or BASIC programs which have been LISTed to a disk or tape. These components are explained in detail within this manual.

For a description of the various files included on the supplied diskette, refer to the "*System Information*" section of this manual.

About the DOS

Draper Pascal can be used with most popular Disk Operating Systems. It has been tested with Atari DOS 2.5, SpartaDOS 3.2d, and MYDOS. You should format a diskette with DOS on it to contain the Draper Pascal system. Since the Draper Pascal Supervisor is named AUTORUN.SYS, it will execute immediately after the disk is booted. For XL and XE computers, you do not need to hold down the Option key while booting unless you are using SpartaDOS. If using SpartaDOS, you may want to rename AUTORUN.SYS to PASCAL.COM and create a STARTUP.BAT file containing the following two lines:

```
BASIC OFF
PASCAL
```

The Shareware Concept

Draper Pascal is distributed on a Shareware basis.

You may freely copy Draper Pascal for distribution under the Shareware concept, without charge.

You may NOT charge any fee for the Draper Pascal program or documentation without our written approval.

You may NOT distribute Draper Pascal or it's documentation in connection with ANY commercial venture, product, publication or service unless you read, sign, and send in the royalty-free license included with this manual.

Ramdisk support

Draper Pascal supports the use of the "Ramdisk" capability provided by using a DOS that supports a ramdisk like Atari DOS 2.5 or SpartaDOS 3.x with an Atari computer system having sufficient memory to support the ramdisk. While using this feature, the Editor takes less than two seconds to load and the Compiler takes less than three seconds.

Diskette preparation for Ramdisk support

To utilize the ramdisk support, you must make sure the DOS on your Draper Pascal diskette has everything in place to create the ramdisk. For example, with Atari DOS 2.5 make sure the disk also contains RAMDISK.COM. For SpartaDOS 3.x, you will need RD.COM (or RD260.COM). These programs are provided with your Disk Operating System (DOS).

To activate the Ramdisk feature for Draper Pascal 2.1, use your version of DOS to rename the following three files. Consult your DOS manual if you need instruction on how to do the rename.

Rename	From this name:	To this name:
	RAMDISK1.DAT	COPYFILE.OBJ
	RAMDISK2.DAT	COPYLIST.TXT
	RAMDISK3.DAT	RAMDISK.PCD

Using the Ramdisk feature

To use the Ramdisk feature, do the following:

1. Boot your diskette and initialize the ramdisk. With Atari DOS 2.5, this would be done automatically if RAMDISK.COM is present on the diskette at boot time. With SpartaDOS 3.x, you must execute RD.COM (or RD260.COM) specifying D8: as the drive number for the ramdisk. If you wish to use a drive number other than 8, you must first edit COPYLIST.TXT and change the second line to contain the desired drive number. With SpartaDOS 3.x, ramdisk intialization could be done automatically by adding RD D8: to the batch file STARTUP.BAT mentioned above.

2. Start Draper Pascal.

3. Enter '1', for Run Program, followed by the name RAMDISK (since RAMDISK.PCD is to be executed). If an Error 138 occurs, it indicates that the ramdisk drive has not been properly initialized. An Error 170 might occur if one of the files being copied to the ramdisk is not found. This could happen if you renamed AUTORUN.SYS to PASCAL.COM as mentioned in "*About the DOS*", above. If this is the case, verify that each file name contained within COPYLIST.TXT is spelled correctly.

That's all there is to it. Your default drive will be set to the ramdisk drive number. This means that if you edit, run, or compile a program and don't specify a particular drive number (Dx:), the default will be assumed.

The RAMDISK program works as follows:

1. The source (input) disk drive number is read from file COPYLIST.TXT.

2. The target (output) disk drive (ramdisk) number is read from file COPYLIST.TXT.

3. A check is made to see if the ramdisk already contains one of the programs to be copied. If so, processing continues with step 6, below.
4. A machine language fast file copying subroutine (COPYFILE.OBJ) is loaded into memory.
5. Each remaining record of file COPYLIST.TXT is read and the corresponding file is copied from the source drive to the target drive.
6. The default drive indicator is set to be the target drive number.

The source code for this program (RAMDISK.PAS) and the file copy subroutine (COPYFILE.M65) are printed with the other source code listings in this manual.

Getting Started

This section is intended to show by example how to use the Draper Pascal system. You will edit, compile, and run a sample program. Information displayed by the computer is shown in normal type while responses to be entered by you are shown underlined with dashes (---). To begin with, make sure you have 48K RAM installed and no cartridge in place. Boot the disk now by placing it in disk drive 1 and turning on the power to the Atari computer. After the Supervisor has finished loading, you will see a screen that looks like this:

```
      DRAPER PASCAL
       VERSION 2.1
  1 - Run Program
  2 - Disk Directory
  3 - Compile Program
  4 - Edit a Program
  5 - Exit to DOS
  6 - List a file
  7 - Trace on

      Copyright 1989
      by Norm Draper


  4                                   Select the Editor
  -


      DRAPER SOFTWARE
          EDITOR

  A - Add line(s) at end
  C - Change line(s)
  D - Delete line(s)
  E - Edit a line
  F - Filer menu
  I - Insert before line
  L - List line(s)
  M - Menu
  P - Print line(s)
  Q - Quit
  S - Scan line(s)
  X - Exit to Compiler
A,C,D,E,F,I,L,M,P,Q,S,X,?->F          Select Filer menu
                           -

  A - Append file
  D - Directory list
  L - Load file
  S - Save file
```

```
L
```

Load a file

```
Enter filename  -> SAMPLE1
```

Enter the name of the file to be loaded. The name of the last file edited, compiled, or run will be filled in by the Editor. You may have to overtype it with the name shown.

```
A,C,D,E,F,I,L,M,P,Q,S,X,?->L
```

List the file on the screen

```
Line from ->_
Line to   ->_
```

Just press RETURN for 'Line from' and 'Line to'. This will give a list of the entire program in memory.

```
  1:PROGRAM KALEIDOSCOPE;
  2:VAR I,J,K,W,X:INTEGER;
  3:BEGIN
  4: MAXGRAPH(19);
  5: GRAPHICS(19);
  6: X:=0;
  7: REPEAT
  8: FOR W:=3 TO 50 DO
  9:  BEGIN
 10:    FOR I:=1 TO 10 DO
 11:    BEGIN
 12:     FOR J:=0 TO 10 DO
 13:      BEGIN
 14:       K:=I+J;
 15:       COLOR(J*3/(I+3)+I*W/12);
 16:       PLOT(I+8,K);
 17:       PLOT(K+8,I);
 18:       PLOT(32-I,24-K);
 19:       PLOT(32-K,24-I);
 20:       PLOT(K+8,24-I);
 21:       PLOT(32-I,K);
 22:       PLOT(I+8,24-K);
 23:       PLOT(32-K,I)
 24:      END
 25:    END
 26:  END
 27: UNTIL X=99 (* UNENDING LOOP *)
 28:END.
A,C,D,E,F,I,L,M,P,Q,S,X,?->I
```

Let's insert a comment before line 15.

```
Line -> 15
```

```
 15:       (* MY FIRST EDIT *)
 16: _
```

Enter the data to be inserted when prompted for line 15. Just press RETURN when prompted for line 16. This will terminate insert mode.

```
A,C,D,E,F,I,L,M,P,Q,S,X,?->L
```

List again to verify that the change was made correctly.

```
Line from ->_
```

```
Line to    ->_

   1:PROGRAM KALEIDOSCOPE;
   2:VAR I,J,K,W,X:INTEGER;
   3:BEGIN
   4: MAXGRAPH(19);
   5: GRAPHICS(19);
   6: X:=0;
   7: REPEAT
   8: FOR W:=3 TO 50 DO
   9:  BEGIN
  10:    FOR I:=1 TO 10 DO
  11:    BEGIN
  12:     FOR J:=0 TO 10 DO
  13:      BEGIN
  14:       K:=I+J;
  15:       (* MY FIRST EDIT *)
  16:       COLOR(J*3/(I+3)+I*W/12);
  17:       PLOT(I+8,K);
  18:       PLOT(K+8,I);
  19:       PLOT(32-I,24-K);
  20:       PLOT(32-K,24-I);
  21:       PLOT(K+8,24-I);
  22:       PLOT(32-I,K);
  23:       PLOT(I+8,24-K);
  24:       PLOT(32-K,I)
  25:      END
  26:    END
  27:  END
  28: UNTIL X=99 (* UNENDING LOOP *)
  29:END.
A,C,D,E,F,I,L,M,P,Q,S,X,?->F
```

Let's save the program back to disk drive 1 under the same name.

```
        A - Append file
        D - Directory list
        L - Load file
        S - Save file

        S

Enter filename   -> SAMPLE1

A,C,D,E,F,I,L,M,P,Q,S,X,?->X
```

Now let's exit directly to the Compiler.

```
         Draper Software

         Pascal Compiler

           Version 2.1

         Copyright 1989
         by Norm Draper


         Enter Filename:
         SAMPLE1
```

Enter name of program to be compiled. The name of the last program edited, compiled, or run will be filled in by the Compiler.

```
Enter List Output Filespec
```

```
Default is E:_
```

Just press RETURN at this point to have the compile list directed to the screen.

```
0000 PROGRAM KALEIDOSCOPE;
0000 VAR I,J,K,W,X:INTEGER;
0003 BEGIN
0003  MAXGRAPH(19);
0017  GRAPHICS(19);
001B  X:=0;
001E  REPEAT
0022  FOR W:=3 TO 50 DO
002A   BEGIN
0035     FOR I:=1 TO 10 DO
003D     BEGIN
0048      FOR J:=0 TO 10 DO
004F       BEGIN
005A        K:=I+J;
0062        COLOR(J*3/(I+3)+I*W/12);
008A        PLOT(I+8,K);
0098        PLOT(K+8,I);
00A6        PLOT(32-I,24-K);
00B8        PLOT(32-K,24-I);
00CA        PLOT(K+8,24-I);
00DC        PLOT(32-I,K);
00EA        PLOT(I+8,24-K);
00FC        PLOT(32-K,I)
010A       END
010A     END
010C   END
011C  UNTIL X=99 (* UNENDING LOOP *)
0142 END.
0147
ADDR   NAME
---- --------
0003 I
0004 J
0005 K
0006 W
0007 X
5 Compiler table entries used
*** Program Execution Completed ***
Highest Stack Address Used = $AFF8
<START>Repeat,<SELECT>Menu,<ESC>Exit
```

Press the SELECT key at this point to take us to the main menu.

```
           DRAPER PASCAL

            VERSION 2.1

        1 - Run Program

        2 - Disk Directory

        3 - Compile Program

        4 - Edit a Program

        5 - Exit to DOS

        6 - List a file

        7 - Trace on


           Copyright 1989
           by Norm Draper
```

1 -	Select '1' to run the program that was just compiled.
Enter name of program to be run	
SAMPLE1 -------	The name of the last program edited, compiled, or run will be filled in by the main menu program. Overtype the name if you want to run a different program.

==

At this point you should have a nice kaleidoscope pattern being displayed on your television screen. To stop it, press the BREAK key. To repeat execution, press the START key. To return to the main menu, press the SELECT key. To exit to DOS, press the ESC key.

Another program, SAMPLE2, is also provided for you to practice with. It will display Roman numerals for powers of two between 1 and 4096. Compile it, turn on the trace via the main menu, and run it. After it is finished, press CTRL-T to display the trace table, and CTRL-S to display the stack contents. When prompted for 'Where? Filespec', enter 'E:'. For a description of the stack display line, refer to the 'DUMPSTK' command in the 'Pascal Definitions' section of the manual provided to registered users.

Main Menu

The Main Menu is the initial program to be run by the Supervisor. It is written in Pascal. The source code is provided for it and you may customize it as you see fit. The disk filename for the source is 'INIT.PAS'. The pseudo code program that is initially executed is 'INIT.PCD'. It would be wise to copy 'INIT.PCD' to another name to be used in case your compile of the menu program is not successful. Or, you could rename INIT.PAS to something else, like NEWINIT.PAS, and compile it to produce NEWINIT.PCD. Then you can use the 'run' option (mentioned below) to test your modified program.

The Main Menu appears as follows:

DRAPER PASCAL

VERSION 2.1

1 - Run Program

2 - Disk Directory

3 - Compile Program

4 - Edit a Program

5 - Exit to DOS

6 - List a file

7 - Trace on

Copyright 1989
by Norm Draper

Each of the menu options will now be explained:

1 - Run Program

Use this option to execute a program that has previously been successfully compiled. You will see the following prompt:

Enter name of program to be run

The Main Menu program will fill in the name of the last program edited, compiled, or run. If this is the one you want, all you have to do is press RETURN. If it is not the one you want, just overtype the name shown with the one you want.

2 - Disk Directory

This option will provide you with a list of all, or selected, files on one of your disk drives. You will receive the prompt 'Filespec?'. If you just press RETURN at this point, you will see a list of all files on the default drive. If you enter 'D2:', you will see all files on drive 2. To show only selected files, use wildcards in the normal manner. For example, enter 'D1:INIT.*' to show only files named INIT with any suffix from drive one. At the end of the list, you will be prompted to press any key to continue. After pressing any key, the Main Menu will be re-displayed.

3 - Compile Program

This option sends you directly to the Pascal compiler. You will be prompted for the name of the program to be compiled, after the Compiler is loaded. If you have already edited, compiled, or run a program, the name will be shown and may be used by just pressing the RETURN key. For more information, refer to the section of this manual on 'The Compiler'.

4 - Edit a Program

Control is transfered to the Draper Pascal Editor when this option is chosen. For more information, refer to the section of this manual on "*The Editor*".

5 - Exit to DOS

Pascal execution is terminated by this option. Control is passed to the Disk Operating System.

6 - List a file

This convenience entry is provided to allow you to view, on the screen, any text file on disk or tape. You are prompted to enter the name of the file to be listed. The file is assumed to reside on the default drive if a colon (:) is not found within the name you specify. At the end of the list, you will be prompted to press any key to continue. After pressing a key, the Main Menu will appear again.

7 - Trace on

The wraparound internal trace may be turned on (or off) with this option. The trace is used only for debugging purposes and may be viewed at program termination time by pressing CTRL-T. Program execution speed is slightly degraded while the trace is active. You will be prompted to enter the number of trace entries to be maintained by the system. Each trace entry requires 10 bytes of storage at the high end of memory. The trace may not be used during graphics displays because screen memory is also at the high end of memory. To turn the trace off and remove the memory allocation of the trace table, enter zero when prompted for the number of entries to maintain. The trace format is described in the "*System Information*" section of this manual.

The Editor

The Editor is used to create, modify, and save Pascal source files. It may also be used to process other text type files, like BASIC programs which have been LISTed to disk or tape. It is a line oriented editor. Combined with some type of formatting program, it may be used for word processing applications. The entire source to be edited must be in memory at one time. If your Pascal program will not fit within the limits of the Editor, then you can use the INCLUDE feature of the Compiler to allow segments of a program to be edited separately. Refer to the section on *"The Compiler"* for more information on the INCLUDE feature. Source code for the Editor is listed under *"Editor Program Source Listing"* in this manual. Some key points to be noted about this editor are as follows:

1. Each line is referred to by line number, however, no line numbers are stored either internally or on the disk or tape.
2. Each line may contain up to 80 characters. This may be changed by altering the constant called MAXLENGTH and re-compiling the Editor. A source listing of the Editor is provided to registered users.
3. A maximum of 250 lines of text may be edited at one time. This may be changed by altering the constant called MAXLINES and re-compiling the Editor. An increase in MAXLINES should correspond with a decrease in MAXLENGTH, and vice versa. A source listing of the Editor is provided to registered users.
4. When entering or editing a line, the line must be terminated by pressing the RETURN key.
5. As lines are inserted into, or deleted from, the source file, the remaining lines are automatically renumbered.
6. A line of source may extend onto more than one screen line.
7. Due to operation of the Atari operating system, a blank line may not be directly entered. To enter a blank line, you must first enter a non-blank character (like a period), then use the Editor Change command to change the character to a space.
8. Input operations (Append and Insert) are terminated by entering a null line (just pressing the RETURN key).
9. The BREAK key is disabled by the Editor to prevent loss of data. It is enabled again at termination of the Editor.
10. If you enter or change data then try to Quit or exit to the Compiler without first saving the data onto disk, you will receive an option to either save the data or ignore it and continue.
11. Cassette tape files may be loaded, edited, and saved by the Editor. The Compiler does not support tape input, though. You would first have to load the file from tape, with the Editor, then save it to disk.

EDITOR COMMANDS

General Prompts

The following prompts are general in nature and are common among many of the editor commands to be described below.

Line ->

You are prompted to enter one line number, as opposed to a range of line numbers. It is used by the INSERT Editor command and refers to the line before which the inserted line(s) will be placed.

Line from ->

This is the first prompt for a range of line numbers. Enter the low number of the range. If you just press RETURN, line number 1 is assumed.

Line to ->

Enter the high line number in the range desired. If only one line is to be acted upon, that number must be entered in both this prompt and the one mentioned above. If you just press RETURN, the highest line number in the buffer will be assumed. If the number you enter is less than the 'Line from' value, the 'Line from' value will be used here.

Enter filename ->

This prompt is shown when loading, appending, and saving files. The last filename used is filled in after the arrow. If this is the file you wish to use now, then all you have to do is press RETURN. A full filespec may be entered, but is not required. If a colon (:) is not found within the filename specified, then the default drive is assumed. If the filename given does not contain a period (.), then a suffix of .PAS is assumed.

The Commands

A - Add line(s) at end

This command is used to add lines after the last line currently in the buffer. If the buffer is currently empty, then line 1 will be assumed as the starting point. In this manner, you can create a new file if one has not been loaded. You can append as many lines as you like. When you are finished entering lines, just press RETURN without entering any data on the line (null line).
Prompts used: None

C - Change line(s)

The Change command allows you to change one specified string pattern to another for the first occurance in each line within the range of lines specified. After being prompted for the line number range, you are asked for the data to 'Change from ->' and 'Change to ->'. Enter any string of characters at each prompt. Imbedded blanks are allowed. If you just press RETURN for the 'Change to' prompt, the first occurance of the 'Change from' data within each line will be deleted.
Prompts used: 'Line from', 'Line to', 'Change from', 'Change to'

D - Delete line(s)

This command allows you to delete a line or a range of lines from the file in memory. The whole file in memory will be deleted if you just press RETURN when prompted for both 'Line from' and 'Line to'. Be aware that all lines following the range deleted will be renumbered, to fill the gap just made. If you desire to delete a number of line ranges, delete those with the highest numbers first and proceed toward the beginning of the file. That way, you won't have to do a LIST after each range delete to find out what the new line numbers for the following lines are.
Prompts used: 'Line from', 'Line to'

E - Edit line(s)

The Edit command is used to edit (or make individual changes to) a line or range of lines that already exist in memory. If a range is specified, the lines are presented to you one at a time. As each line is presented, you may use any of the normal Atari editing keys (like right and left cursor, insert, delete), to alter the data. Just press RETURN when you are finished with each change. If you don't want to make a change to a line shown, just press RETURN.
Prompts used: 'Line from', 'Line to'

F - Filer menu

The Filer is a subsystem which handles communication with an external device (disk or tape). The features provided are as follows:

A - Append file

A file is read from disk or tape and added to the end of the file currently in memory. The data in memory prior to the append remains unchanged.
Prompts used: 'Enter filename'

D - Directory list

This command is used to provide a directory list of the different files on a diskette. You are prompted for 'Filespec?'. Enter the disk drive number and selection criteria for the directory list. If you just press RETURN you will see a directory list of all files on the default drive. To see all files on drive two, enter 'D2:' or 'D2:*.*'. To see only files with a suffix of PAS on drive one, enter 'D1:*.PAS'.
Prompts used: 'Filespec?'

L - Load file

This is the way to load a file into memory from disk or tape. If any data was currently in memory, it is deleted and replaced by the file read in.
Prompts used: 'Enter filename'

S - Save file

Data is copied from memory to disk or tape with this command. The data currently in memory remains unchanged. You are prompted for filename and may use whatever name you wish. It is not necessary to save a file under the same name as was used to load the file. You should save data to disk frequently if you are making extensive changes. That way you won't have to re-do as much if something goes wrong.
Prompts used: 'Enter filename'

I - Insert before line

This command allows you to insert one or more lines at any point within the file in memory. The inserted data is placed before the line number you specify. To terminate insert mode, just press RETURN without entering any data on the same line (null line). Note that all lines after the point of insertion will automatically be renumbered.
Prompts used: 'Line ->'

L - List line(s)

One or more lines of data from memory are listed on the screen with this command. During the list, you may stop the scrolling by pressing either the space bar or RETURN. To resume scrolling, press any other key other than ESC. The ESC key may be pressed to prematurely terminate the listing.
Prompts used: 'Line from', 'Line to'

M - Menu

The main Editor menu is presented in response to this command. A question mark (?) may also be used to display the main menu.
Prompts used: None

P - Print line(s)

This command is used to create a list of data in memory on a printer attached to the Atari parallel port (P:). Internal line numbers are also directed to the printer although they do not actually exist within the file on disk or tape.
Prompts used: 'Line from', 'Line to'

Q - Quit

This command is used to exit from the Editor when you are finished editing your data. Control is given to the Main Menu program. If you have changed the data in memory and have not saved it prior to quitting, you will be given the option of saving the data or ignoring the changes and exiting. If you are going to compile a Pascal program immediately after quitting the Editor, you may use the 'X' command described below.
Prompts used: None

S - Scan line(s)'

This command allows you to display all lines within a specified range which contain a specified character string. The character string may contain any characters, including imbedded blanks. To temporarily stop the listing, press either the space bar or RETURN. To abort the listing, press ESC. Press any other key to continue as normal.
Prompts used: 'Line from', 'Line to', 'Scan for'

X - Exit to Compiler

This command terminates the Editor and transfers control directly to the Compiler. If the file in memory has been changed but not saved prior to the Exit command, you will be prompted to either save the file or ignore the changes and proceed to the Compiler.
Prompts used: None

The Compiler

The Compiler is used to translate words that we humans understand into "words" that the computer can understand. The computer words are referred to as pseudo-code, or p-code for short. These pseudo-code instructions are understood and executed by the Supervisor.

This is a single pass goal oriented compiler. It expects the proper syntax for a statement. If correct syntax is not found, the compilation stops, and an error number with associated text description is displayed. At this point, you are given the option of quitting or returning to the Editor to correct the problem and do the compile again.

The Compiler itself is written in Draper Pascal and occupies about 28K of RAM memory space.

The first prompt from the Compiler is 'Enter filename:'. The name of the last program edited, run, or compiled is filled in for your convenience. If this is the one you want, just press RETURN. If it is not the one you want, just overtype it with the name you desire. The name you provide will become the new default name for the Editor, Compiler, and Main Menu 'Run' option. No suffix is allowed when specifying filename. The Compiler will add the standard '.PAS' to it for you. If the source does not reside on the default disk drive, then you must prefix the filename with 'Dn:' where 'n' is the disk drive number where the source resides. The default disk drive is normally disk drive number one, but is changed to the Ramdisk drive number if you are taking advantage of the Ramdisk feature of a Disk Operating System that supports it. Ramdisk initialization is explained in the manual provided to registered users.

The next prompt is 'Enter List Output Filespec'. The default (if you just press RETURN) is the screen (E:). The list output may go to any normal output device, such as printer (P:) or disk (D:LISTNAME.PRN).

A number of additional points are mentioned below:

1. Comments are delimited by '(*' on the left end and '*)' on the right end. Any characters may appear within comments. Comments may appear anywhere within the program.

2. 'Include' files are supported. You may have procedures, functions, or any part of a program included in a compile, even though it is not actually part of the file being compiled. It is a variation of a comment which allows you to do this. The format is as follows:

   (*$I XXXXXXXX *) or (*$I D1:XXXXXXXX *)

   The dollar sign and 'I' must be right next to '(*' and must be followed by one space. Then you may mention the 'D' for disk and drive number (if other than the default drive is to be used). Follow it with a colon (:) and the filename. A suffix of '.PAS' will be automatically added to the file name. Then have at least one space and '*)'.

3. Pascal source files must reside on disk.

4. The output pseudo-code from the compile will be directed to the same disk drive that the Pascal source resides on. It will be created with a filename suffix of '.PCD'. If you have multiple disk drives and the source and pcode will not both fit on one disk, have a small file on the output disk with an 'include' for the source which resides on the other disk.

5. The hexadecimal offset of the pseudo instructions generated is

given at the left side of the output listing. This offset may be useful for debugging purposes. It may be referred to when looking at a program trace (see TRACEON in the Pascal Definitions section of the manual provided to registered users). It also may be referred to in case of an error message or termination caused by pressing the BREAK key. The offset shown may not always be accurate. If not exact, the values are very close.

6. The name and stack offset of each variable defined is shown at the end of the compile listing. The offset value is shown in hexadecimal. Each stack entry is two bytes wide. The first three stack entries are reserved for system use. Therefore, the offset of the first variable will be 0003, which is actually six bytes into the stack. If a variable is defined within a procedure or function, the offset shown is relative the beginning of that procedure or function.

7. The program is ready to run immediately after the compile is finished. No linking is required. (Some Pascal systems require linking of output code after the compile and before execution).

8. Nested procedures are supported. You may define one procedure within another.

9. Recursive procedures are supported. A procedure may call itself. If variables are defined within the procedure, they are cleared with each entry into the procedure and refreshed upon exit from the recursive procedure call.

10. No forward references are allowed. A procedure may not be referenced before it is defined. In most cases, nesting the procedures will take care of this problem.

11. Double density disk drives are supported for both source and pcode files. The pcode will be written to the same drive that the initial source is taken from.

12. Only integer type parameters may be passed to procedures and functions. Other types of data may be passed by using global type variables setup at the beginning of the program (not within a procedure or function).

13. A function may only return an integer type value. Procedures do not return values.

14. Hexadecimal constants and literals are prefixed by dollar signs ($).

15. To write out an integer in hexadecimal format, precede the variable name with a percent sign (%).

16. A total of 170 compiler table entries may be used. One table entry is used for each variable definition, procedure name, function name, and parameter name used with procedures and functions. Table entries for variables defined within procedures are re-used following the 'END' for that procedure. The number of table entries used within a compile is displayed at the end of the output list from the Compiler.

17. The time needed to compile a program can be reduced by turning off the ANTIC chip within the computer. This turns off the display to the screen yet gives a fairly significant increase to the Atari's internal speed. In a normal Pascal program, you can have POKE(559,0) to turn it off and POKE(559,34) to turn it back on. But a special compile time option is provided to make use of this feature to speed up compiles. It is as follows. Have a statement (*$S+*) to turn the ANTIC off (increase speed), and use (*$S-*) to turn the ANTIC on (resume

normal speed). These options may appear anywhere within a program. The ANTIC is automatically turned back on at compile termination and at time of error (if any).

The Supervisor

The Supervisor is a high performance machine language program which simulates a pseudo 16-bit stack oriented computer. It executes the pseudo code that is generated by the Compiler.

It is loaded into memory by disk operating system at the hex location $1D7C, which is just above DOS in memory. It should work with any DOS that allows a program to load at that address, such as Atari DOS 2.1S, Atari DOS 2.5, or SpartaDOS version 2.x or higher. A message will be displayed if the Supervisor cannot be loaded at the proper location.

The disk filename for the Supervisor's object code is 'AUTORUN.SYS'. It may be renamed to anything you desire, such as 'PASCAL.COM', but will not be automatically loaded when the disk is booted if the name is other than 'AUTORUN.SYS'. To start the Pascal system from the DOS menu, use the 'L', binary load, option to load 'AUTORUN.SYS' into memory. Execution will begin automatically.

The Supervisor begins execution by loading and executing the Pascal program 'INIT.PCD' from the default drive, which is always disk drive 1 immediately after loading the Supervisor. 'INIT.PCD' is the name of the main menu program. You may substitute any compiled Pascal program of your own by naming it 'INIT.PCD'. In this manner, you can have a true turnkey system where your program begins execution after booting the disk.

After termination of each Pascal program, the Supervisor gives you a choice of what to do next. You are prompted with the following line:

<START>Repeat,<SELECT>Menu,<ESC>Exit

If you press the START key, your Pascal program will execute again from the beginning. If you press the SELECT key, control will be transfered to the main menu program (INIT.PCD). If you press the ESC key, you will exit to the DOS utility menu. You also have two other options at this point. They are both used for debugging purposes. If you press CTRL-S (the 'S' key while holding down the CTRL key), the stack values, at termination time, will be displayed. If you press CTRL-T, the internal trace table, if active, will be displayed. With either of these two debugging options, you will be asked where the display should be sent by the prompt 'WHERE? (FILESPEC)'. To see it on the screen, enter 'E:'. It also may be sent to printer or disk by following normal filespec naming conventions. If the display is sent to the screen, you may stop the scrolling by use of the space bar. Press the ESC key if you have seen enough and wish to return to the Supervisor termination prompt. Any other key causes scrolling to continue as normal.

Pascal Definitions

ABS FUNCTION ABS(Number):INTEGER;

This function returns the absolute value of 'Number'. In effect, all it does is return the value of 'Number' with a positive sign. 'Number' may be any integer expression.

Example:
```
PROGRAM ABS_DEMO;
VAR AJ,J:INTEGER;

BEGIN
  J:=-7;
  AJ := ABS(J);
  WRITELN('ABS OF -7 IS ',AJ)
END.
```

BASIC Equivalent: AJ = ABS(J)

ADDR FUNCTION ADDR(Var):INTEGER;

This function returns the integer absolute address of the specified variable. The variable may be of any type. If it is an element of an array, the address returned is that of the particular element specified. For a description of the data formats, see the item titled 'Internal Data Formats' in the 'System Information' section of this manual.

Example:
```
PROGRAM ADDR_DEMO;
VAR A,B:INTEGER;

BEGIN
  A := ADDR(B);
  WRITELN('ADDRESS OF B IS ',A)
END.
```

BASIC Equivalent: A = ADR(J$) (Applies only to string variable in Atari BASIC)

AND

This operator sets the resulting condition as true if both the left and right factors around it are true, otherwise, the condition is set to false. Parentheses should surround the factors on each side.

Example:
```
PROGRAM AND_DEMO;
VAR A:INTEGER;

BEGIN
  IF (A>0) AND (A<7) THEN
    WRITELN('VALUE WITHIN RANGE')
END.
```

BASIC Equivalent: Same as Pascal

ARCTAN FUNCTION ARCTAN(Var):REAL;

ARCTAN is a REAL built-in function that returns the value of an angle whose tangent is equal to the value of the variable specified. 'Var' may be either a REAL variable or an INTEGER variable, but the value returned is always REAL.

Example:
```
PROGRAM ARCTAN_DEMO;
VAR R1,R2:REAL;

BEGIN
  WRITELN('Enter a number');
  READ(R1);
  R2:=ARCTAN(R1);
  WRITELN('The ARCTAN of ',R1,' is ',R2)
END.
```

BASIC equivalent: R2=ATN(R1)

ARRAY

ARRAY[Number1] OF Type
ARRAY[Number1,Number2] OF Type

ARRAY specifies that multiple occurances of a variable are to be defined. Either one or two dimension arrays may be defined. For single dimension arrays, 'Number2' and the comma that precedes it must be omitted. 'Number1' and 'Number2' may be either integer numbers or previously defined integer constants. They specify the number of elements to be dimensioned. For two dimension arrays, 'Number1' represents the number of rows, while 'Number2' represents the number of columns within each row. Space is reserved for 'Number'+1 entries because occurance numbers of zero through 'Number' are allocated. This means that ARRAY[2] defines space for three entries, numbered 0, 1, and 2. ARRAY[2,3] defines space for twelve entries; rows 0 through 3 with four columns (0 through 3) in each row. While using an array, note that the index for the element in the array, which is specified within parentheses '()', must either be an integer number or an integer type variable.

Examples:
```
PROGRAM ARRAY_DEMO;
CONST SIZE=4;
VAR I,ROW,COL:INTEGER;
    A1: ARRAY[3] OF INTEGER;
    A2: ARRAY[SIZE] OF STRING;
    A3: ARRAY[2,3] OF INTEGER;

BEGIN
  FOR I:=0 TO 3 DO
    A1(I):=I;
  FOR I:=0 TO SIZE DO
    A2(I):='';
  FOR ROW:=0 TO 2 DO
    FOR COL:=0 TO 3 DO
      A3(ROW,COL):=ROW+COL;
END.
```

BASIC Equivalent: DIM A(3)
No equivalent for BASIC string variables.

ASC

FUNCTION ASC(Cvar):INTEGER;

This function returns the ASCII value (integer) of the specified character variable.

Example:
```
PROGRAM ASC_DEMO;
VAR I:INTEGER;
    CH:CHAR;

BEGIN
```

```
        CH:='A';
        I := ASC(CH);
        WRITELN('THE ASCII VALUE OF ',CH,' IS ',I)
      END.
```

BASIC Equivalent: I = ASC(CH)

BEGIN

BEGIN marks the start of a block or compound statement within a Pascal program. END marks the termination of the block or compound statement. Each statement between the BEGIN and the END, except for the last one, should be followed by a semicolon (;).

Example: PROGRAM BEGIN_DEMO;

```
      BEGIN
        WRITELN('My name is Fred');
        WRITELN;
        WRITELN
      END;
```

BASIC Equivalent: None

BLOAD PROCEDURE BLOAD(Program);

This exclusive built-in procedure loads the specified program (or data) from disk into memory. The program to be loaded should be in the standard DOS load format as generated by an appropriate assembler or the binary save function of DOS. 'Program' should be specified in the normal filespec format, including extension, if any. The object loaded will not automatically begin execution after completion of the load, as some programs do. The machine language program will be executed by use of the CALL built-in procedure. Refer to the CALL description for further information. The IORESULT value should be checked after the BLOAD to verify that the program did, in fact, exist on the disk.

Explanation for example:
The Pascal program below sends the ASCII value of each of the upper case letters to the 6502 assembler subroutine. The subroutine changes the character to inverse and then changes it into a lower case character before returning control to the Pascal program. The Pascal program then retrieves the character from the subroutine, prints it on the screen, and repeats until the alphabet is complete.

Example: PROGRAM BLOAD_DEMO_1;

```
      VAR I:INTEGER;
          CH:CHAR;

      BEGIN
        OPTIONS(0);
        BLOAD('D:TEST.OBJ');
        OPTIONS(1);
        IF IORESULT <> 0 THEN
          WRITELN('TEST.OBJ NOT ON DISK');
        FOR I:=ASC('A') TO ASC('Z') DO
          BEGIN
            POKE($600,I);
            CALL($601);
            CH:=PEEK($600);
            WRITE(CH)
          END;
        WRITELN
```

```
END.
```

```
*** 6502 Assembler subroutine used in above demo
10   *=$600
20 ADDR1 .BYTE 0
30   LDA ADDR1 Get character from Pascal
40   ORA #$80  Make character inverse
50   CLC       Prepare for add instruction
60   ADC #32   Make character lower case
70   STA ADDR1 Put back character for Pascal
80   RTS       Return to Pascal program
90   .END
```

The capability is also provided for the accumulator, the X register, and the Y register to be initialized for the machine language programs use. The value for the accumulator should be stored into memory location 166 ($A6). The initial values for the X and Y registers go into locations 167 and 168 ($A7 and $A8) respectively. When control is returned to the Pascal program, the ending values of the accumulator, X register, and Y register may be found in these same locations. Using this technique, the same demo program could be made up as follows:

Example:
```
PROGRAM BLOAD_DEMO_2;
VAR I:INTEGER;
    CH:CHAR;

BEGIN
  OPTIONS(0);
  BLOAD('D:TEST.OBJ');
  OPTIONS(1);
  IF IORESULT <> 0 THEN
    WRITELN('TEST.OBJ NOT ON DISK');
  FOR I:=ASC('A') TO ASC('Z') DO
    BEGIN
      POKE($A6,I);
      CALL($600);
      CH:=PEEK($A6);
      WRITE(CH)
    END;
  WRITELN
END.
```

```
*** 6502 Assembler subroutine used in above demo
10   *=$600
20   ORA #$80  Make character inverse
30   CLC       Prepare for add instruction
40   ADC #32   Make character lower case
50   RTS       Return to Pascal program
60   .END
```

BASIC Equivalent: None, however some BASIC programs POKE machine language programs into memory after READing the ASCII values for each byte of the program as contained in DATA statements.

BOOLEAN

BOOLEAN is a type code which can represent one of two states, TRUE of FALSE. The actual value is either zero for FALSE or one for TRUE. A BOOLEAN variable can be used to save the result of a condition.

Example:
```
PROGRAM BOOLEAN_DEMO;
VAR ANSWER:BOOLEAN;
```

```
BEGIN
  ANSWER:=TRUE;
  ANSWER:=FALSE;
  ANSWER:= X < 0;
  ANSWER:= (X < 0) OR (X > 99)
END.
```

BASIC Equivalent: None

CALL PROCEDURE CALL(Address);

The CALL procedure transfers execution to a machine language program at the specified address. 'Address' is any integer expression, which includes hex constants. It is equivalent to the assembler operation JSR (jump to subroutine). The subroutine should return control to the Pascal program by using the RTS (return from subroutine) operation. No parameters are passed to the subroutine directly, so the 6502 stack will not be loaded with a number of parameters, as is done by Atari BASIC. This simply means that the machine language subroutine should not have a PLA (pull accumulator) instruction at its start as is customary with machine language subroutines called from an Atari BASIC USR instruction. If the subroutine does begin with PLA and no parameters are being passed, you can just have the call refer to the address of the byte after the PLA instruction. However, the accumulator, the X register, and the Y register may be initialized before a call to the subroutine and inspected after returning from the subroutine. Refer to the explanation under BLOAD for more details.

Example: Refer to BLOAD example

BASIC Equivalent: None, but quite similar to the USR instruction, as mentioned above.

CASE

```
CASE expr1 OF const1 : stmt1;
              const2 : stmt2;
              ...
              constn : stmtn
END;

CASE expr1 OF const1 : stmt1;
              const2 : stmt2;
              ...
              constn : stmtn
ELSE stmtx
END;
```

The CASE statement compares the result of an expression with several constants to determine the appropriate statement to be executed.

Example:

```
PROGRAM CASE_DEMO;
VAR DAY:INTEGER;

BEGIN
  WRITE('Enter day number ');
  READ(DAY);
  CASE DAY OF
    1 : WRITELN('Monday');
    2 : WRITELN('Tuesday');
    3 : WRITELN('Wednesday');
    4 : WRITELN('Thursday');
    5 : WRITELN('Friday');
```

```
      6 : WRITELN('Saturday');
      7 : WRITELN('Sunday')
    ELSE
      WRITELN('Invalid day number')
  END
END.
```

BASIC Equivalent: None

CHAR

This is a type code assigned to variables to be used in character format. For the reading of character type variables, one character of data is transfered from the input device to the variable. No carriage return (RETURN) is required to terminate the input.

Example:
```
PROGRAM CHAR_DEMO;
VAR CH:CHAR;

BEGIN
  READ(CH);
  CASE CH OF
    'A' : WRITELN('First letter');
    'B' : WRITELN('Last letter')
  END
END.
```

BASIC Equivalent: None.

CHR FUNCTION CHR(expr1):CHAR;

This function changes an integer value into a character format. 'expr1' may be any integer expression. If the value of 'expr1' is greater than 255, then the ASCII value of the character value returned will be 'expr1' modulo 256. CHR must be used if it is desired to write a character which is not a normal letter or number, such as sending control codes to a printer or clearing the screen. The CHR(125) in the following example is the proper code for clearing the screen.

Example:
```
PROGRAM CHR_DEMO;
VAR CH:CHAR;
    I:INTEGER;

BEGIN
  WRITE('Enter a number between 0 and 255 ');
  READ(I);
  CH:=CHR(I);
  WRITELN(CHR(125),'Character equivalent is ',CH)

END.
```

BASIC Equivalent: CH=CHR$(I)

CLOSE PROCEDURE CLOSE(File);

This built-in procedure closes a previously opened file. 'File' may either be a variable of type FILE, or an absolute IOCB number, such as #1. It does not hurt to close a file which is already closed. Multiple files may be specified if separated by commas.

Example: Refer to examples for EOF and EOLN

BASIC Equivalent: CLOSE #2

COLOR PROCEDURE COLOR(Number);

This built-in procedure determines the data to be stored in the display memory for all subsequent PLOT and DRAWTO built-in procedures. It's purpose is identical to that of the COLOR command in BASIC. Please refer to your Atari BASIC manual for further information. 'Number' may be any integer expression.

Example: Refer to example for GRAPHICS

BASIC Equivalent: COLOR 2

CONCAT PROCEDURE CONCAT(Parm1,Parm2,...):STRING;

This built-in function returns a string value equal to the concatenation of all parameters specified in the CONCAT function. These parameters may be of type string constant, string variable, or character variable.

Example:
```
PROGRAM CONCAT_DEMO;
VAR PGMNAME:STRING;
BEGIN
  WRITE('Enter file name ');
  READLN(PGMNAME);
  PGMNAME := CONCAT(PGMNAME,'.TXT');
END.
```

BASIC Equivalent: PGMNAME$(LEN(PGMNAME$+1))='.TXT'

CONST CONST name1=value1; name2=value2; ...

CONST is used to declare constants to be used within a program. The value of a constant cannot be changed. The values may be of type integer or real. String constants are not permitted. The most efficient method for simulating string constants is to declare space for them with the VAR declarative, then read in the values from a disk file. Hexadecimal integers may be defined by preceding the value with a dollar sign ($).

Example:
```
PROGRAM CONST_DEMO;
CONST NUMTIMES = 4; PI = 3.1416;
      ACCUM = $A6;
VAR I:INTEGER;
    RADIUS,ANSWER:REAL;

BEGIN
  FOR I:=1 TO NUMTIMES DO
    BEGIN
      WRITE('Enter radius ');
      READ(RADIUS);
      ANSWER := PI * (RADIUS * RADIUS);
      WRITELN('Circumference is ',ANSWER)
    END
END.
```

BASIC Equivalent: None

COPY FUNCTION COPY(Source,Index,Length) : STRING;

This built-in function returns a string value composed of a portion of the string named by 'Source'. The portion consists of 'Length' characters starting at offset 'Index' into 'Source'. The first position of a string has the index value

of 1. 'Index' and 'Length' are integer expressions, while 'Source' must be of type string. 'Length' must not be negative and must have a value in the range 1-255. The same is true for 'Index'. If the value of 'Index' plus 'Length' is greater than the length of 'Source', then 'Length' assumes the value of the length of 'Source' minus 'Index'.

Example:
```
PROGRAM COPY_DEMO;
VAR FULL_NAME,LAST_NAME:STRING;
    I:INTEGER;

BEGIN
  FULL_NAME := 'SMITH, JOHN B';
  I := POS(',',FULL_NAME);
  LAST_NAME := COPY(FULL_NAME,1,I-1);
  WRITELN('The last name of ',FULL_NAME,
          ' is ',LAST_NAME)
END.
```

BASIC Equivalent: A$=B$(4,7)

COS FUNCTION COS(Var):REAL;

COS is a built-in function which returns the cosine of the value of the variable 'Var'. 'Var' may be either an INTEGER variable or a REAL variable. The value returned will always be a REAL value.

Example:
```
PROGRAM COS_DEMO;
VAR R1,R2:REAL;


BEGIN
  WRITELN('Enter a real number');
  READ(R1);
  R2:=COS(R1);
  WRITELN('The cosine of ',R1,' is ',R2)
END.
```

BASIC equivalent: R2=COS(R1)

CVTREAL FUNCTION CVTREAL(Ivar):REAL

This built-in function can be used to copy the value of an INTEGER variable into a REAL variable. 'Ivar' must be an INTEGER type variable.

Example:
```
PROGRAM CVTREAL_DEMO;
VAR I1:INTEGER;
    R1:REAL;


BEGIN
  WRITELN('Enter an integer number');
  READ(I1);
  R1:=CVTREAL(I1);
  WRITELN(R1,' is now a real number')
END.
```

BASIC Equivalent: None

DEG PROCEDURE DEG;

DEG is used to specify that the output values from ARCTAN, COS, and SIN are to be expressed in degrees, as opposed to

radians. The system defaults to radians unless DEG is specified. Once specified, all output is in degrees until RAD is specified for radians, or the computer is turned off and back on.

Example:
```
PROGRAM DEG_RAD_DEMO;
VAR R1,R2:REAL;
    REPLY:CHAR;

BEGIN
  WRITELN('Enter a D for output in degrees');
  WRITELN('      or R for output in radians');
  READ(REPLY);
  CASE REPLY OF
    'D': DEG;
    'R': RAD
  ELSE
    WRITELN('That was not one of the choices')
  END;
  WRITELN('Enter a real number');
  READ(R1);
  R2:=SIN(R1);
  WRITELN('The sine of ',R1,' is ',R2)
END.
```

BASIC Equivalent: DEG

DELETE PROCEDURE DELETE(Source,Index,Size);

The DELETE built-in procedure removes a specified number of characters from a string. 'Size' characters are removed from the string, 'Source', starting at offset 'Index'.

Example:
```
PROGRAM DELETE_DEMO;
VAR ALPHABET:STRING;

BEGIN
  ALPHABET:='ABCDEFG';
  DELETE(ALPHABET,3,2);
  WRITELN(ALPHABET)
END.
```

The resulting value of ALPHABET will be 'ABEFG'.

BASIC Equivalent: None

DIV

This operator computes the quotient of the two factors surrounding it. The factors may be either of type REAL or type INTEGER. DIV is equivalent to '/' in this implementation of Pascal.

Example:
```
PROGRAM DIV_DEMO;
VAR I1,I2:INTEGER;
    R1,R2,R3:REAL;

BEGIN
  I1:=20;
  I2:=I1 DIV 2;
  R1:=20.0;
  R2:=5.2;
  R3:=R1 DIV R2
END;
```

BASIC Equivalent: R3=R1/R2

DOS PROCEDURE DOS;

This built-in procedure terminates execution of the Pascal supervisor and transfers control to the Atari Disk Operating System. For more information on the use of DOS, refer to the DOS Manual.

Example:
```
PROGRAM DOS_DEMO

BEGIN
  DOS
END.
```

BASIC Equivalent: DOS

DRAWTO PROCEDURE DRAWTO(X,Y);

The DRAWTO built-in procedure causes a graphic line to be drawn from the last coordinate refered to in a PLOT or DRAWTO built-in procedure. The color of the line is determined by the most recent setting of the COLOR procedure. 'X' and 'Y' may be any valid integer expressions.

Example:
```
PROGRAM DRAWTO;
VAR X,Y:INTEGER;

BEGIN
 COLOR(1);
 PLOT(10,10);
 X:=20;
 Y:=30;
 DRAWTO(X,Y)
END;
```

BASIC Equivalent: DRAWTO X,Y

DUMPSTK PROCEDURE DUMPSTK;

This exclusive built-in procedure dumps the values of the Pascal stack to the output device of your choice. The output is sent to IOCB #7. If it is already open, then it will be used as is. If it is not open, the following prompt will be displayed on the screen: 'WHERE? (FILESPEC)'. Enter with a normal device specification, such as E:. Each stack entry is two bytes wide. It is displayed in the following format:

STACK ADDR=aaaa HEX=hhhh CHAR=cc

'aaaa' is the absolute address of this stack entry, shown in hexadecimal format. 'hhhh' is the value of this stack entry shown in hex. 'cc' is the same stack entry value shown in character format if the value is determined to be printable. Refer to the 'System Information' section of this manual for a description of internal variable formats.

Example:
```
PROGRAM DUMPSTK_DEMO;

BEGIN
  DUMPSTK
END.
```

BASIC Equivalent: None

DVSTAT PROCEDURE DVSTAT(A,B,C,D);

This exclusive built-in procedure reads the device status information as requested from the STATUS command and stores the values into variables 'A', 'B', 'C', and 'D'. These variables may have any names, but must be predefined as integer variables. The values stored into the named variables are taken from locations 746 through 749, decimal, within the operating system. The most common usage for DVSTAT would be in checking the status of RS232 ports. Consult your Atari 850 Interface Module Operator's Manual for the meanings associated with these different status bytes.

Example:
```
PROGRAM DVSTAT_DEMO;
VAR BYTE1,BYTE2,BYTE3,BYTE4:INTEGER;

BEGIN
  STATUS(#1);
  DVSTAT(BYTE1,BYTE2,BYTE3,BYTE4);
  WRITELN('Status values are ',
    BYTE1,' ',
    BYTE2,' ',
    BYTE3,' ',
    BYTE4)
END.
```

BASIC Equivalent:
```
A=PEEK(746)
B=PEEK(747)
C=PEEK(748)
D=PEEK(749)
```

END

END marks the termination of a block or compound statement within a Pascal program. BEGIN marks the start of the block or compound statement. Each statement between the BEGIN and the END, except for the last one, should be followed by a semicolon (;). END is also required as termination for a CASE statement.

Example: Refer to example for BEGIN.
BASIC Equivalent: None

EOF EOF(File);

This reserved word checks for end of file of an input device. It returns a true value if the most recent read of the file has detected an end of file mark. 'File' may be either a variable of type FILE, or an absolute IOCB number preceded by a '#'.

Example:
```
PROGRAM EOF_DEMO;
VAR INPUT,OUTPUT:FILE;
    DATA:STRING;

BEGIN
  RESET(INPUT,'D:TEST.TXT');
  REWRITE(OUTPUT,'D:TEST.NEW');
  REPEAT
    READLN(INPUT,DATA);
    WRITELN(OUTPUT,DATA)
  UNTIL EOF(INPUT);
  CLOSE(INPUT,OUTPUT)
END.
```

BASIC Equivalent:
```
100 TRAP 2000
...
2000 IF PEEK(195)=136 THEN ...
```

EOLN EOLN(File);

This reserved word checks for end of line of an input device. It returns a true value if the most recent read of the file has detected an end of line condition ($9B character). 'File' may be either a variable of type FILE, or an absolute IOCB number preceded by a '#'.

Example:
```
PROGRAM EOLN_DEMO;
VAR DATA:CHAR;

BEGIN
  OPEN(#1,4,0,'D:TEST.TXT');
  OPEN(#2,8,0,'D:TEST.NEW');
  REPEAT
    READ(#1,DATA);
    WRITE(#2,DATA);
    IF EOLN(#1) THEN WRITELN(#2);
  UNTIL EOF(#1);
  CLOSE(#1,#2)
END.
```

BASIC Equivalent:
```
100 GET #1,A
200 IF A=155 THEN ...
```

EXIT PROCEDURE EXIT;

This built-in procedure causes immediate termination of the currently executing Pascal program. Control is transfered to the Pascal Supervisor. No files are closed.

Example:
```
PROGRAM EXIT_DEMO;

BEGIN
  EXIT
END.
```

BASIC Equivalent: END

EXP FUNCTION EXP(Var):REAL;

The function EXP(Var) computes the value of e to the 'Var' power. 'Var' may be either an INTEGER variable or a REAL variable. The value returned is always a REAL number. e is the base of the natural logarithm. The exponential function (EXP) and the natural logarithmic function (LN) are inverse functions.

Example:
```
PROGRAM EXP_DEMO;
VAR R1,R2:REAL;

BEGIN
  R1:=3.0;
  R2:=EXP(R1)
END.
```

BASIC equivalent: R2=EXP(R1)

EXP10 FUNCTION EXP10(Var):REAL;

The function EXP10(Var) computes the value of 10 to the 'Var' power. 'Var' may be either an INTEGER variable or a REAL variable. The value returned is always a REAL number. The exponential function (EXP10) and the decimal logarithmic function (LOG) are inverse functions.

Example:
```
PROGRAM EXP10_DEMO;
VAR R1,R2:REAL;

BEGIN
  R1:=3.0;
  R2:=EXP10(R1)
END.
```

BASIC equivalent: R2=10 ^ R1

FALSE

FALSE is a BOOLEAN constant representing the untrue state. It is internally equal to an integer value of zero.

Example: Refer to the example under BOOLEAN
BASIC Equivalent: None

FILE

This is a type code used in a VAR declaration. Each file defined is internally assigned an IOCB number. These numbers start at one, for the first file defined, and increment up to a maximum value of seven. The FILE type variables may only be used in input-output type commands such as OPEN, CLOSE, READ, READLN, WRITE, WRITELN, EOF, EOLN, RESET, and REWRITE.

Example: (Refer to example under EOF)

BASIC Equivalent: None

FOR

```
FOR var := expr1 TO expr2 DO statement;
FOR var := expr1 DOWNTO expr2 DO statement;
```

The FOR statement is used to repeat execution of a statement for a predefined number of times. 'var' and 'expr1' and 'expr2' must be of the same type. The types allowed are INTEGER and REAL. Execution is as follows:

1. 'var' is set to 'expr1'.
2. 'var' is compared with 'expr2'.
   If 'var' is greater than or equal to 'expr2' (for TO)
   or 'var' is less than or equal to 'expr2' (for DOWNTO)
   proceed to step 6.
3. 'statement' is executed.
4. 'var' is incremented by 1 (for TO)
   or decremented by 1 (for DOWNTO).
5. go to step 2.
6. exit

Example:
```
PROGRAM FOR_TEST;
VAR I:INTEGER;

BEGIN
  FOR I:=1 TO 5 DO WRITELN('TEST')
END.
```

BASIC Equivalent: FOR I=1 TO 5

FUNCTION

A FUNCTION is a group of statements that has a name and executes a certain task or algorithm. The identifier name for the FUNCTION may be used as a variable of type INTEGER. Parameters may be passed to the FUNCTION. These parameters must also be of type INTEGER. In this implementation of Pascal, FUNCTION may be abbreviated as FUNC.

Example:
```
PROGRAM FUNCTION_TEST;
VAR A,B:INTEGER;

FUNCTION SQUARE(NUMBER);

BEGIN
  SQUARE:=NUMBER*NUMBER
END;

BEGIN (*MAIN*)
  FOR A:=1 TO 5 DO
    BEGIN
      B:=SQUARE(A);
      WRITELN('THE SQUARE OF ',A,' IS ',B)
    END
END.
```

BASIC Equivalent: None

GOTOXY PROCEDURE GOTOXY(X,Y);

This built-in procedure is used to set the position of the cursor. The next WRITE will have it's output begin at x-coordinate 'X' and y-coordinate 'Y'. The cursor will not actually be moved until the next WRITE occurs. 'X' and 'Y' can be any integer expressions.

Example:
```
PROGRAM GOTOXY_TEST;

BEGIN
  GOTOXY(12,12);
  WRITELN('MIDDLE OF SCREEN')
END.
```

BASIC Equivalent: POSITION 12,12

GRAPHICS PROCEDURE GRAPHICS(Number);

The GRAPHICS command is used to select one of the many graphics modes available on the Atari computer. For a complete description of the command and the modes available, please refer to your Atari BASIC manual. 'Number' may be any integer expression. Note that before using the GRAPHICS command, you should execute the MAXGRAPH command to reserve screen memory for the mode desired. If you don't, the Pascal stack may overlay part of the screen memory and the results would be unpredictable.

Example:
```
PROGRAM KALEIDOSCOPE;
VAR I,J,K,W:INTEGER;

BEGIN
  MAXGRAPH(19);
  GRAPHICS(19);
  X:=0;
  REPEAT
    FOR W:=3 TO 50 DO
      BEGIN
```

```
                    FOR I:=1 TO 10 DO
                      BEGIN
                        FOR J:=0 TO 10 DO
                          BEGIN
                            K:=I+J;
                            COLOR(J*3/(I+3)+I*W/12);
                            PLOT(I+8,K);
                            PLOT(K+8,I);
                            PLOT(32-I,24-K);
                            PLOT(32-K,24-I);
                            PLOT(K+8,24-I);
                            PLOT(32-I,K);
                            PLOT(I+8,24-K);
                            PLOT(32-K,I)
                          END
                      END
                  END
              UNTIL KEYPRESS
              END.
```

BASIC Equivalent: GRAPHICS 8

HIMEM PROCEDURE HIMEM(Value);

This built-in procedure is used to set the upper boundary of memory to be used by the Pascal supervisor during execution. 'Value' may be any integer expression. HIMEM may be used to protect a machine language subroutine in upper memory, or to protect an area of memory where you may store data.

Example:
```
PROGRAM HIMEM_DEMO;

BEGIN
  HIMEM($5FFF);
  BLOAD('D:TEST.OBJ');
  CALL($6000)
END.
```

BASIC Equivalent: POKES into locations 144 and 145 (decimal)

IF
```
IF expr1 THEN stmt1;
IF expr1 THEN stmt1 ELSE stmt2;
```

The IF statement evaluates expressions to see if they are true or false. 'expr1' is any kind of expression. If the expression is true, then 'stmt1' will be executed. If the expression is false, then 'stmt1' is not executed. If ELSE is used then 'stmt2' is executed when the expression is false.

Example:
```
PROGRAM IF_DEMO;
VAR I:INTEGER;

BEGIN
  I:=5;
  IF I=5 THEN WRITELN('FIVE')
         ELSE WRITELN('NOT FIVE')
END.
```

BASIC Equivalent: IF I=5 THEN ... (No ELSE)

INSERT PROCEDURE INSERT(Source,Destination,Index);

This built-in procedure inserts a string, or string literal, into another string at a specified position. 'Source' may be either a string variable, a string literal (within quotes), or a character type variable. 'Destination' must be a

variable of type string. 'Index' may be any integer expression having a value in the range 1-255.

Example:
```
PROGRAM INSERT_DEMO;
VAR PGMNAME:STRING[20];

BEGIN
  WRITE('Enter filename ');
  READLN(PGMNAME);
  IF POS(':',PGMNAME) = 0 THEN
    INSERT('D1:',PGMNAME,1);
  WRITELN('New filename is ',PGMNAME)
END.
```

BASIC Equivalent: None

INTEGER

INTEGER is a type code assigned to integer variables. Integer variables contain values which are whole numbers in the range -32768 to +32767.

Example: Refer to ASC example

BASIC Equivalent: None

IORESULT FUNCTION IORESULT:INTEGER;

The IORESULT built-in function returns the value of the return code from the most recent input-output operation. It is normally used after disk operations to verify that the requested action successfully completed. If the value of IORESULT is zero, then the operation was successful. If it is other than zero, some kind of error has occurred. End-of-file and end-of-line are not considered errors and are handled by the EOF and EOLN built-in functions. An integer variable may be assigned the value of IORESULT if the value is to be saved. Remember that WRITE and WRITELN cause input-output operations to occur and set the value of IORESULT. Refer to the BASIC or ASSEMBLER manuals for a list of the error codes and their meanings. The error numbers above 127 are the ones you should be concerned with. The value of 137 (truncated record) may pertain to some of the built-in string functions and not actually be caused by an input-output request.

Example: Refer to BLOAD example

BASIC Equivalent: The TRAP instruction is used to provide a line number to branch to on error conditions.

KEYPRESS FUNCTION KEYPRESS:INTEGER;

This built-in function returns a one (true value) if any key on the keyboard has been pressed. Otherwise the value returned is a zero (false value). It allows a program to continue executing until interrupted by someone pressing a key on the keyboard.

Example: Refer to example under GRAPHICS

BASIC Equivalent: IF PEEK(764)<>255 THEN ...

LENGTH FUNCTION LENGTH(svar):INTEGER;

The LENGTH built-in function returns the length of a string. 'svar' must be a string type variable.

Example:
```
PROGRAM LENGTH_DEMO;
VAR I:INTEGER;
    S:STRING;

BEGIN
  S:='ABCDEFG';
  I:=LENGTH(S);
  WRITELN('The length of ',S,' is ',I)
END.
```

BASIC Equivalent: I=LEN(S$)

LN FUNCTION LN(Var):REAL;

The LN function returns the natural logarithm of the value of 'Var'. 'Var' mey either be an INTEGER variable or a REAL variable, but must be positive and greater than zero. The value returned will always be REAL.

Example:
```
PROGRAM LN_DEMO;
VAR R1,R2:REAL;


BEGIN
  R1:=3.0;
  R2:=LN(R1)
END.
```

BASIC Equivalent: R2=LOG(R1)

LOCATE FUNCTION LOCATE(X,Y):INTEGER;

The LOCATE function positions the invisible graphics cursor at the specified location in the graphics window and returns a value equal to the data at that pixel. Graphics modes 0 through 2 will return a value of 0-255. The 2-color graphics modes will return a value of 0 or 1. The four color modes will return a value in the range 0-3. You should reposition the cursor using GOTOXY prior to doing a WRITE after LOCATE.

Example:
```
PROGRAM LOCATE_DEMO;
VAR I,X,Y:INTEGER;

BEGIN
  MAXGRAPH(19);
  GRAPHICS(19);
  SETCOLOR(2,8,10);
  PLOT(8,12);
  DRAWTO(12,12);
  I:=LOCATE(10,12);
  GRAPHICS(0);
  WRITELN('The data was ',I)
END.
```

BASIC Equivalent: LOCATE 10,12,I

LOCK PROCEDURE LOCK(Filename);

LOCK is used to lock a file on disk. After a file is locked, it is protected from being accidentally deleted or renamed. 'Filename' may either be a string literal (in quotes) or a string type variable.

Example:
```
PROGRAM LOCK_DEMO;
VAR FILENAME:STRING;

BEGIN
  FILENAME:='D:TEST.TXT';
  LOCK(FILENAME);
  LOCK('D:TEST.TXT')
END.
```

BASIC Equivalent: XIO 35,#1,0,0,"D:TEST.TXT"

LOG FUNCTION LOG(Var):REAL;

The LOG function returns the decimal logarithm (to the base 10) of the value of 'Var'. 'Var' may be either an INTEGER variable or a REAL variable. The value of 'Var' must be positive. The value returned will always be REAL.

Example:
```
PROGRAM LOG_DEMO;
VAR R1,R2:REAL;

BEGIN
  R1:=3.0;
  R2:=LOG(R1)
END.
```

BASIC Equivalent: R2=LOG(R1)/LOG(10)

LPENH, LPENV FUNCTION LPENH:INTEGER;
FUNCTION LPENV:INTEGER;

These two functions are used for light pen support. LPENH returns the horizontal (X-coordinate) of the light pens position, while LPENV returns the vertical (Y-coordinate) position.

Example:
```
PROGRAM LPEN_DEMO;
VAR A,X,Y:INTEGER;

BEGIN
  MAXGRAPH(8);
  GRAPHICS(8);
  COLOR(1);
  REPEAT
    IF SELECTKEY THEN GRAPHICS(8);
    WHILE STICK(0)=15 DO
      BEGIN
        X:=LPENH;
        Y:=LPENV;
        PLOT(X,Y);
        IF IORESULT<>0 THEN EXIT
      END
  UNTIL KEYPRESS
END.
```

BASIC Equivalent: X=PEEK(564):Y=PEEK(565)

MAXGRAPH PROCEDURE MAXGRAPH(Mode);

The MAXGRAPH procedure is used to inform Pascal of the maximum graphics mode to be used within the program. Internal pointers are adjusted to allow for the required amount of screen memory to be reserved. If MAXGRAPH is not used, you may get undesirable results if the internal stack overlays part of the screen memory. 'Mode' may be any valid graphics

mode, including those with 16 or 32 added to them. If the internal trace (see TRACEON) is active, it is forced off by the MAXGRAPH command.

Example: Refer to example under LPENH

BASIC Equivalent: None

MOD

MOD is an operator used to compute the remainder after the division of two integer factors. The left factor is divided by the right factor with the value returned being the remainder of the division.

Example:
```
PROGRAM MOD_DEMO;
VAR I,YEAR:TNTEGER;

BEGIN
  WRITELN('Enter year ');
  READ(YEAR);
  I:=YEAR MOD 4;
  IF I=0 THEN
    WRITELN('Leapyear')
  ELSE
    WRITELN('Not leapyear')
END.
```

BASIC Equivalent: None

NOT

This is an operator used to complement the factor which follows it. It is most commonly used to determine when to stop reading input (WHILE NOT EOF DO ...).

Example: Refer to example under EOF

BASIC Equivalent: NOT

NOTE PROCEDURE NOTE(Iocbno,Sector,Byte);

The NOTE procedure is used to retrieve and save the current access location of a disk file. 'Iocbno' may be any valid IOCB number which refers to an open disk file. The IOCB number should be preceded by a #. 'Byte' and 'Sector' refer to previously defined integer type variables. NOTE and POINT are used together to provide random access to disk files.

Example:
```
PROGRAM NOTE_POINT_DEMO;
VAR SECTOR,BYTE,I,REPLY:INTEGER;
    S_TABLE,B_TABLE:ARRAY[5] OF INTEGER;
    DATA:STRING;

BEGIN

  (* CREATE THE FILE *)

  OPEN(#1,8,0,'D:TEST.TXT');
  FOR I:=1 TO 5 DO
    BEGIN
      WRITELN('Enter record number ',I);
      READLN(DATA);
      NOTE(#1,SECTOR,BYTE);
      S_TABLE(I):=SECTOR;
      B_TABLE(I):=BYTE;
      WRITELN(#1,DATA)
```

```
      END;
    CLOSE(#1);

    (* RANDOMLY ACCESS THE FILE *)

    OPEN(#1,4,0,'D:TEST.TXT');
    FOR I:=1 TO 5 DO
      BEGIN
        REPEAT
          WRITE('Enter a record number ');
          READ(REPLY)
        UNTIL (REPLY>0) AND (REPLY<6);
        SECTOR:=S_TABLE(REPLY);
        BYTE:=B_TABLE(REPLY);
        POINT(#1,SECTOR,BYTE);
        READLN(#1,DATA);
        WRITELN('Record ',REPLY,' is ');
        WRITELN(DATA)
      END;
    CLOSE(#1)
  END.
```

BASIC Equivalent: NOTE

ODD FUNCTION ODD(iexp);

The ODD function returns a true value if the value of the specified integer expression is odd. 'iexp' may be any integer type expression.

Example:

```
PROGRAM ODD_DEMO;
VAR I:INTEGER;

BEGIN
  WRITE('Enter an integer number ');
  READ(I);
  IF ODD(I) THEN
    WRITELN('Odd')
  ELSE
    WRITELN('Even')
END.
```

BASIC Equivalent: None

OPEN PROCEDURE OPEN(Fileno,Aux1,Aux2,Filename);;

The OPEN is used to connect a program to a device. Each device or file must be opened before it may be accessed. The RESET and REWRITE commands may also be used to open files. 'Fileno' may either be a variable of type FILE, or an absolute IOCB number preceded by a #. 'Filename' may be either a variable of type string, or a string literal (within quotes). 'Aux1' specifies the type of open to be performed. Valid values for 'Aux1' are as follows:

- 4 : Input operation
- 6 : Disk directory input operation
- 8 : Output operation
- 9 : End of file append operation
- 12 : Input and output operation

'Aux2' is a device dependant value but is normally zero. Refer to the appropriate manuals for information on specific control codes.

Example: Refer to the example for NOTE

BASIC Equivalent: OPEN #1,4,0,"D:TEST.TXT"

OPTIONKEY

This special built-in function returns a true value if the OPTION key on the Atari keyboard is being pressed at the time the instruction is executed.

Example:
```
PROGRAM OPTIONKEY_DEMO;
VAR I:INTEGER;

BEGIN
  WRITELN('Press BREAK key to stop');
  REPEAT
    IF OPTIONKEY THEN WRITELN('Option key');
    IF SELECTKEY THEN WRITELN('Select key');
    IF STARTKEY THEN WRITELN('Start key')
  UNTIL I=99 (* UNENDING LOOP *)
END.
```

BASIC Equivalent:
```
IF PEEK(53279)=4 THEN ... :REM OPTION KEY
IF PEEK(53279)=2 THEN ... :REM SELECT KEY
IF PEEK(53279)=1 THEN ... :REM START KEY
```

OPTIONS OPTIONS(Opt1,Opt2,...,Optn);

This special built-in procedure allows you to control certain events at program execution time. The options specified are always integer numbers. They are defined in pairs so that one number can set an option while the other number of the pair can reset the same option. An option remains in effect until reset by the other option in the pair, or the Pascal Supervisor is reloaded. The 'S' on the end of the word OPTIONS is required, even if only one option number is specified. If an invalid option number is given, it will be ignored and execution will continue as normal. The available options are shown below with defaults shown:

0 - TURN OFF ERROR DISPLAY
The display of CIO error messages is suppressed with this option. Error conditions can be checked for by looking at the value of IORESULT after each input-output operation.

1 - TURN ON ERROR DISPLAY (Default)
This option allows CIO error messages to be displayed when they occur.

2 - TURN OFF PROMPT DISPLAY
This option suppresses the printing of the 'Execution Completed' message and the 'Highest Stack Address Used' message.

3 - TURN ON PROMPT DISPLAY (Default)
This option allows the above mentioned messages to be once again displayed at program termination.

4 - DISABLE BREAK KEY
This option prevents the BREAK key on the Atari keyboard from interrupting execution of a program. In order to keep the BREAK key disabled, it may be necessary to have OPTIONS(4) specified after the first WRITE or WRITELN that goes to the screen or any OPEN, RESET, or REWRITE that addresses the screen (E: or S:). It should also be reissued after the GRAPHICS command.

5 - ENABLE BREAK KEY (Default)
The BREAK key may once again be used to stop

execution of a program after this option is put into effect.

6 - ONLY POSITIVE INTEGERS (0 TO 65535)
This option sets the range of integer values to be from zero through 65535. Reads, writes, and compares are affected by the setting of this option.

7 - POSITIVE AND NEGATIVE INTEGERS (-32768 TO +32767)
This option sets the range of integer values to be from -32768 through +32767. Reads, writes, and compares are affected by the setting of this option.

Example:
```
PROGRAM OPTION_DEMO;
VAR REPLY:CHAR;

BEGIN
  WRITELN('Enter D to disable break key');
  WRITELN('Enter E to enable break key');
  READ(REPLY);
  CASE REPLY OF
  'D' : OPTIONS(4);
  'E' : OPTIONS(5)
END.
```

BASIC Equivalent: None

OR

This operator sets the resulting condition as true if either the left or the right factors around it are true, otherwise, the condition is set to false. Parentheses should surround the factors on each side.

Example:
```
PROGRAM OR_DEMO;
VAR A:INTEGER;

BEGIN
  WRITE('Enter a number between 1 and 6');
  READ(A);
  IF (A<1) OR (A>6) THEN
    WRITELN('Value outside of range')
   ELSE
    WRITELN('Value okay')
END.
```

BASIC Equivalent: Same as Pascal

ORD FUNCTION ORD(Realvar):INTEGER;

The ORD function is used to convert a real number into an integer number. 'Realvar' must be a variable of type REAL. Rounding, rather than truncation, is performed on the value. Refer to the example for a method of obtaining a truncated value.

Example:
```
PROGRAM ORD_DEMO;
VAR I:INTEGER;
    R:REAL;

BEGIN
  WRITE('Enter a real number ');
  READ(R);
  I:=ORD(R);
  WRITELN('The rounded integer value is ',I);
```

```
        IF CVTREAL(ORD(R)) > R THEN
          R:=R-1;
        WRITELN('The truncated value is ',ORD(R))
      END.
```

BASIC Equivalent: I=INT(R)

PADDLE FUNCTION PADDLE(Number):INTEGER;

This function returns the status value of a particular paddle controller. The controllers are numbered 0-7 from left to right. The value returned will be an integer number between 1 and 228. The value increases as the knob on the controller is rotated counterclockwise. 'Number' may be any integer expression having a value in the range 0-7.

Example:

```
PROGRAM PADDLE_DEMO;
VAR I,J:INTEGER;

BEGIN
  REPEAT
    I:=PADDLE(0);
    WRITELN('Value of paddle(0) is ',I)
  UNTIL J=99 (* UNENDING LOOP *)
END.
```

BASIC Equivalent: I=PADDLE(0)

PEEK FUNCTION PEEK(Address):INTEGER;

This function returns the contents of a specific memory address location. The value returned will be an integer in the range 0-255. 'Address' may be any integer expression, including hexadecimal constants (preceded by a $).

Example:

```
PROGRAM PEEK_DEMO;
VAR I,REPLY:INTEGER;

BEGIN
  WRITE('Enter a memory address in decimal');
  READ(REPLY);
  I:=PEEK(REPLY);
  WRITELN('That location contains hex ',%I)
END.
```

BASIC Equivalent: I=PEEK(REPLY)

PLOT PROCEDURE PLOT(X,Y);

PLOT is used to display a point within one of the graphics windows. The color of the point plotted is determined by the hue and luminance in the color register from the last COLOR statement executed. The color of the plotted point is changed by use of the SETCOLOR command. 'X' and 'Y' may be any integer expressions.

Example: Refer to example under GRAPHICS

BASIC Equivalent: PLOT(X,Y)

POINT PROCEDURE POINT(Iocbno,Sector,Byte);

The POINT procedure is used to position the disk file pointer to the next location to be read or written. It is used in conjunction with NOTE to provide random access capabilities. 'Iocbno' may be any valid IOCB number which refers to an open

disk file. It must be preceded by a '#'. 'Sector' and 'Byte' refer to previously defined integer type variables. They normally contain a value which was set by a NOTE command.

Example: Refer to example under NOTE

BASIC Equivalent: POINT #1,SECTOR,BYTE

POKE PROCEDURE POKE(Address,Value);

The POKE procedure is used to store a certain value into a specific memory location. 'Address' may be any integer expression, including hexadecimal constants (preceded by a $). 'Value' may be any integer expression. 'Value' should be in the range 0-255. If it is greater than 255, then the value stored will be 'Value' MOD 256.

Example:
```
PROGRAM POKE_DEMO;
CONST LEFT_MARGIN = 82;
VAR I:INTEGER;

BEGIN
  WRITE('Enter new left margin value ');
  READ(I);
  POKE(LEFT_MARGIN,I)
END.
```

BASIC Equivalent: POKE 82,I

POS FUNCTION POS(Pattern,Source):INTEGER;

This function returns the position of the first occurence of a given string in another string. 'Pattern' may be either string variables, character variables, or string literals (within quotes), or any mixture thereof. 'Source' must be a string variable. A value of zero is returned if the pattern is not found. You can easily check for the presence or absence of a pattern by checking to see if the value returned is zero or not.

Example: Refer to example under INSERT

BASIC Equivalent: None

PROCEDURE PROCEDURE Name;
PROCEDURE Name(Parm1,Parm2,...,Parmn);

A procedure is a named group of statements that executes a specific task or algorithm. No value is associated with it, as with a function. Parameters may be passed to the procedure. All parameters must be of type integer. A procedure is activated just by specifying it's name. It must be defined before it's name is mentioned. Variables may be defined within procedures. If they are, they are local to that procedure and may be referenced only from within that procedure. The variable names may be the same as variables defined elsewhere within the program without interfering with their values. In this implementation of Pascal, you may use PROC as an abbreviation for PROCEDURE.

Example:
```
PROGRAM PROCEDURE_DEMO;
VAR NUMLINES:INTEGER;

(* WRITE VARIABLE NUMBER OF BLANK LINES *)

PROCEDURE LINES(NUMBER);
VAR I:INTEGER;
```

```
BEGIN
  FOR I:=1 TO NUMBER DO WRITELN
END;

(* DISPLAY MENU LIST *)

PROCEDURE MENU;

BEGIN

  (* THE 125 BELOW IS A CLEAR SCREEN CODE *)

  WRITELN(CHR(125),'TITLE');
  WRITELN('1 - Choice one');
  WRITELN('2 - Choice two')
END;

(* MAIN PROGRAM SECTION *)

BEGIN
  MENU;
  WRITE('Enter number of lines to blank ');
  READ(NUMLINES);
  LINES(NUMLINES)
END.
```

BASIC Equivalent: The object of a GOSUB

PROGRAM PROGRAM Name;

PROGRAM is used to give a name to the Pascal program which follows it. No code is generated from it. It's only purpose is to provide documentation. 'Name' may be any string of characters, of any length, which is terminated by a semicolon (;).

Example: PROGRAM ANY_NAME_AT_ALL;

```
BEGIN
  WRITELN('This program has a name')
END.
```

BASIC Equivalent: None

PTRIG FUNCTION PTRIG(Number):INTEGER;

This function is used to determine the status of the trigger button on the designated paddle controller. A value of 0 is returned if the trigger is pressed, otherwise the value returned is a 1.

Example: PROGRAM PTRIG_DEMO;

```
BEGIN
  REPEAT
    WRITELN('Press paddle 0 trigger to stop')
  UNTIL PTRIG(0)=0
END.
```

BASIC Equivalent: IF PTRIG(0)=0 THEN ...

PURGE PROCEDURE PURGE(Filespec);

This procedure is used to remove a file from a diskette. 'Filespec' may be either a string variable or a string

literal (within quotes). 'Filespec' must indicate the device and filename extension (if present).

Example:
```
PROGRAM PURGE_DEMO;

BEGIN
  PURGE('D:TEST.TXT')
END.
```

BASIC Equivalent: XIO 33,#1,0,0,"D:TEST.TXT"

RAD

RAD is used to indicate that the output from all trigonometric computations that follow is to be expressed in radians, rather than degrees. Radians are the default unless DEG is specified. You can switch back and forth between degrees and radians as often as you like.

Example: Refer to example under DEG

BASIC Equivalent: RAD

READ, READLN PROCEDURE READ(File,Var1,Var2,...Varn);

READ and READLN are used to supply data to a program from a keyboard or any other input type device. In this implementation of Pascal, READ and READLN are identical and may be used interchangeably. Variables must be predefined to hold the data to be read. These variables may be of type character, integer, real, or string, or elements of an array of one of these types. The type code of the variable determines how it is read into the program. For character type variables, one character of data is transfered from the input device to the variable. No carriage return (RETURN) is required for character type variables. The carriage return is required, however, for all other data types, since each may be entered as a variable number of characters. 'File' is optional, and if present, determines the device from which the data will be read. 'File' may be specified as either an absolute IOCB number (preceded by a #), or a variable of type FILE. If 'File' is not specified, then the Atari keyboard is assumed to be the input device. Any number of variables may be mentioned within a READ statement. 'File' may also be repeated and sets the device to be used as input for each variable that follows it until either another 'File' or the right parenthesis ')' is encountered.

Example: Refer to EOF and EOLN examples

BASIC Equivalent: INPUT #1;VARIABLE

REAL

The REAL type code is used to define variables which are numeric but not integers (contain decimal points) or have values outside the integer range (-32768 through +32767, or 0 through 65535, depending on the setting of option 6 or 7). Each real variable defined occupies three stack positions (six bytes). The format used is identical to that used by BASIC and the Atari operating system. When a real variable is set to a real constant value within a program, the constant must start with an integer, and be followed by a decimal point, and optionally an exponent portion.

Example:
```
PROGRAM REAL_DEMO;
VAR R:REAL;
```

```
BEGIN
  R:=0.55E+3;
  WRITELN('R=',R)
END.
```

BASIC Equivalent: All numeric variables used by Atari BASIC are considered REAL numbers.

RECORD

The RECORD type code is used to define a variable, or group of variables, which are to be read, written, or moved, as an entity in internal format. The variables within the record must be uniquely named and are to be used exactly as if they were not part of a record. The different fields within the record do not have to be all of the same type. All variable types, including arrays, are supported, with the exception of FILE and RECORD. An 'END;' must be present after the last field of the record to indicate the end of the record.

WRITE, rather than WRITELN, should be used when writing records. If WRITELN is used, an end of line character is written following the record and special consideration must be given for it when reading the record back in.

Example: PROGRAM RECORD_DEMO;

```
VAR REC1:RECORD;
      NAME:STRING[20];
      GRADE:REAL;
      AGE:INTEGER;
    END;

    I:INTEGER;
    RECFILE:FILE;

BEGIN
  REWRITE(RECFILE,'D:TEST.REC');
  FOR I:=1 TO 3 DO
    BEGIN
      WRITE('NAME:':10);
      READLN(NAME);
      WRITE('GRADE:':10);
      READLN(GRADE);
      WRITE('AGE:':10);
      READLN(AGE);
      WRITE(RECFILE,REC1)
    END;
  CLOSE(RECFILE);
  RESET(RECFILE,'D:TEST.REC');
  FOR I:=1 TO 3 DO
    BEGIN
      READ(RECFILE,REC1);
      WRITELN('NAME=',NAME);
      WRITELN('AGE=',AGE);
      WRITELN('GRADE=',GRADE)
    END;
  CLOSE(RECFILE)
END.
```

BASIC Equivalent: None.

REPEAT REPEAT Stmt1; ... ;Stmtn UNTIL Condition;

REPEAT is used to loop through a group of statements until a specified condition occurs. The statements are executed at least once, even if the UNTIL condition is initially false.

The condition is tested after the group of statements is executed. 'Condition' may be any normal expression. To test a condition before executing a group of statements, use WHILE.

Example: Refer to example under EOF

BASIC Equivalent: None

RESET PROCEDURE RESET(File,Filespec);

RESET is used to open a file which will be used in input mode. The IOCB is first closed by RESET before the open takes place. 'File' must refer to a variable of type FILE. 'Filespec' refers to the file specifications and may be either a string literal (within quotes) or a string type variable.

Example: Refer to example under EOF

BASIC Equivalent:
```
CLOSE #1
OPEN #1,4,0,"D:TEST.TXT"
```

REWRITE PROCEDURE REWRITE(File,Filespec);

REWRITE is used to open a file which will be used in output mode. The IOCB is first closed by REWRITE before the open takes place. 'File' must refer to a variable of type FILE. 'Filespec' refers to the file specifications and may be either a string literal (within quotes) or a string type variable.

Example: Refer to example under EOF

BASIC Equivalent:
```
CLOSE #1
OPEN #1,8,0,"D:TEST.TXT"
```

RND FUNCTION RND(Iexp):INTEGER;

The RND function is a random number generator. A random integer number is returned between zero and the value of 'Iexp', inclusive. 'Iexp' may be any integer expression.

Example:
```
PROGRAM RND_DEMO;
VAR I1,I2:INTEGER;

BEGIN
  FOR I1:=1 TO 50 DO
    BEGIN
      I2:=RND(25);
      WRITELN(I2)
    END
END.
```

BASIC Equivalent: I2=RND(0)*25

SELECTKEY

This special built-in function returns a true value if the SELECT key on the Atari keyboard is being pressed at the time

the instruction is executed.

Example: Refer to example under OPTIONKEY

BASIC Equivalent: IF PEEK(53279)=2 THEN ...

SETCOLOR PROCEDURE SETCOLOR(Register,Hue,Luminance);

This built-in procedure is used to set the particular hue and luminance to be assigned to a particular color register. 'Register' may be any integer expression which results in a value in the range 0-4. 'Hue' may be any integer expression which results in a value in the range 0-15. 'Luminance' may be any integer expression which results in an even number in the range 0-14. For further information on the SETCOLOR command, refer to the Atari BASIC manual.

Example:
```
PROGRAM SETCOLOR_DEMO;

BEGIN
  MAXGRAPH(3);
  GRAPHICS(3);
  SETCOLOR(0,2,8);
  PLOT(17,1);
  DRAWTO(17,10);
  DRAWTO(9,18);
  PLOT(19,1);
  DRAWTO(19,18);
  PLOT(20,1);
  DRAWTO(20,18);
  PLOT(22,1);
  DRAWTO(22,10);
  DRAWTO(30,18)
END.
```

BASIC Equivalent: Same as BASIC

SHL Expr1 SHL Expr2

The SHL operator performs a bitwise shift of 'Expr1' to the left by 'Expr2' bit positions. Each bit position shifted is equivalent to 'Expr1' multiplied by 2. The value returned is an integer and both 'Expr1' and 'Expr2' refer to integer type expressions. When multiplying an integer by a value which is a power of two, the SHL is more efficient than the multiply (*).

Example:
```
PROGRAM SHL_DEMO;
VAR I,J:INTEGER;

BEGIN
  J:=2;
  I :=J SHL 8;
  WRITELN('2*256=',I)
END.
```

BASIC Equivalent: I=J*(some power of 2)

SHR Expr1 SHR Expr2

The SHR operator performs a bitwise shift of 'Expr1' to the right by 'Expr2' bit positions. Each bit position shifted is equivalent to 'Expr1' divided by 2. The value returned is an integer and both 'Expr1' and 'Expr2' refer to integer type expressions. When dividing an integer by a value which is a power of two, the SHR is more efficient than the divide (DIV or '/').

Example:
```
PROGRAM SHR_DEMO;
VAR I,J:INTEGER;

BEGIN
  J:=1024;
  I :=J SHL 7;
  WRITELN('1024/128=',I)
END.
```

BASIC Equivalent: I=J/(some power of 2)

SIN FUNCTION SIN(Var):REAL;

SIN is a function which returns the sine of the value of 'Var'. 'Var' may be either an INTEGER variable or a REAL variable. The value returned is always REAL.

Example: Refer to the example under DEG

BASIC Equivalent: A=SIN(2)

SOUND PROCEDURE SOUND(Voice,Pitch,Distortion,Volume);

This built-in procedure is used to support the sound capabilities of the Atari computer. 'Voice' refers to one of the four sound registers and may be any integer expression which results in a value 0-3. 'Pitch' is used to set the frequency of the sound. It may be any integer expression which results in a value 0-255. 'Distortion' is used to set the purity of the tone. It may be any integer expression which results in an even number in the range 0-14. A value of 10 creates a pure tone. 'Volume' determines how loud the tone will be played. It may be any integer expression which results in a value 1-15. A value of 1 creates a barely audible sound and a value of 15 creates a loud sound. A value of 0 is used to turn off the sound. For additional information on SOUND, refer to the Atari BASIC manual.

Example:
```
PROGRAM SOUND_DEMO;
VAR I:INTEGER;

BEGIN
  FOR I:=29 TO 121 DO
    BEGIN
      SOUND(0,I,10,10);
      WAIT(15) (* HOLD FOR 1/4 SECOND *)
    END;
  SOUND(0,0,0,0) (* TURN OFF SOUND *)
END.
```

BASIC Equivalent: SOUND (Same as BASIC)

SQR FUNCTION SQR(Var):REAL;

The SQR function returns the square of the value of 'Var'. 'Var' may either be an INTEGER variable or a REAL variable. The value returned will always be REAL.

Example:
```
PROGRAM SQR_DEMO;
VAR R1,R2:REAL;

BEGIN
  R1:=10.0;
  R2:=SQR(R1)
```

```
        END.
```

BASIC Equivalent: R2=R1*R1

SQRT FUNCTION SQRT(Var):REAL;

The SQRT function returns the square root of the value of 'Var'. 'Var' may either be an INTEGER variable or a REAL variable. The value returned will always be REAL.

Example:
```
PROGRAM SQRT_DEMO;
VAR R1,R2:REAL;

BEGIN
  R1:=10.0;
  R2:=SQRT(R1)
END.
```

BASIC Equivalent: R2=SQR(R1)

STARTKEY

This special built-in function returns a true value if the START key on the Atari keyboard is being pressed at the time the instruction is executed.

Example: Refer tp example under OPTIONKEY

BASIC Equivalent: IF PEEK(53279)=1 THEN ...

STATUS PROCEDURE STATUS(Iocbno,Ivar);

This built-in procedure is used to retrieve status information from a particular device. 'Iocbno' refers to either an absolute IOCB number (preceded by a #), or a FILE type variable. 'Ivar' is an INTEGER variable which will contain the return code of the STATUS command. The actual status values returned from the device can be interrogated by using DVSTAT.

Example: Refer to example under DVSTAT.

BASIC Equivalent: STATUS (Same as BASIC)

STICK FUNCTION STICK(Number):INTEGER;

This function returns the status value of a particular joystick attached to the computer. 'Number' refers to the controller jack that the joystick is plugged into. It may be any integer expression which results in a value of 0-3. Values returned for the various positions of the joystick are shown below:

```
        14
   10    |    6
  11-   15   - 7
    9    |    5
        13
```

Example:
```
PROGRAM JOYSTICK_DEMO;
VAR I:INTEGER;

BEGIN
```

```
          REPEAT
            I:=STICK(0);
            WRITELN('Stick 0 is ',I)
          UNTIL KEYPRESS
        END.
```

BASIC Equivalent: I=STICK(0) (Same as BASIC)

STR FUNCTION STR(Var):STRING;

This built-in function is used to convert a number into it's string equivalent. 'Var' may either be an integer type variable or a real type variable.

Example:
```
PROGRAM STR_DEMO;
VAR I:INTEGER;
    R:REAL;
    S:STRING;

BEGIN
  I:=20;
  S:=STR(I);
  WRITELN(S);
  R:=3.1416;
  S:=STR(R);
  WRITELN(S)
END.
```

BASIC Equivalent: S=STR(I)

STRIG FUNCTION STRIG(Number):INTEGER;

This function is used to check on the status of the joystick trigger button. A value of zero is returned if the button is being pressed at the time the instruction is executed. A value of one is returned when the button is not pressed. 'Number' refers to the controller jack that the joystick is plugged into. It may be any integer expression which results in a value 0-3.

Example:
```
PROGRAM STRIG_DEMO;
VAR I:INTEGER;

BEGIN
  REPEAT
    WRITELN('Press button on joystick 0 to stop')
  UNTIL STRIG(0)=0
END.
```

BASIC Equivalent: IF STRIG(0)=0 THEN ...

STRING

STRING is a type code used to define variables which contain a number of characters. A fixed amount of memory is reserved for each string, but the actual length of the string is variable. Any ATASCII codes may be contained within a string variable. String variables may be defined with lengths of 1-255 characters. The length specification is made by putting the length within brackets '[]' after the word STRING. If no length code is specified, a default length of 80 characters is assumed. The functions and procedures used to manipulate strings are CONCAT, COPY, DELETE, INSERT, LENGTH, and POS.

Example:
```
PROCEDURE STRING_DEMO;
VAR A:STRING;
    B:STRING[10];
```

```
            C:ARRAY[5] OF STRING[20];

        BEGIN
          (* 'A' is a string of length 80          *)
          (* 'B' is a string of length 10          *)
          (* 'C' is a six element (0-5) string     *)
          (*     array with each element having    *)
          (*     a length of 20                    *)
        END.
```

BASIC Equivalent: DIM A$(80) No equivalent for string arrays.

TRACEOFF PROCEDURE TRACEOFF;

This special built-in procedure is used to turn off a pseudo instruction code trace that is active if turned on by TRACEON. The wraparound buffer used by the trace is not released by TRACEOFF.

Example: Refer to example under TRACEON.

BASIC Equivalent: None

TRACEON PROCEDURE TRACEON;
PROCEDURE TRACEON('Number);

This special built-in procedure is used to turn on a pseudo instruction trace, for debugging purposes. The trace table is maintained in a memory buffer. 'Number' is used to specify the number of trace entries to maintain. It is a wraparound type trace buffer where new entries overlay old entries if the buffer is not large enough to contain all of the instructions executed. Each trace entry is nine bytes long. The trace entries may be displayed at program termination by entering CTRL-T. Refer to the 'Supervisor' section of this manual for more information. 'Number' may be any integer expression. If 'Number' (and the parentheses) are not specified, then the trace is re-activated using an existing buffer from a previous TRACEON where 'Number' was specified. If the value of 'Number' is zero, then the trace buffer is released from memory and the trace is turned off. Note that the MAXGRAPH command will also turn off the trace and release the memory used for the trace buffer.

Example:
```
PROGRAM TRACE_DEMO;
VAR NAME:STRING;

BEGIN
  TRACEON(100);
  WRITE('Enter your name ');
  READLN(NAME);
  TRACEOFF
END.
```

BASIC Equivalent: None

TRUE

TRUE is a BOOLEAN constant representing the true state. It is internally equivalent to an integer constant of one.

Example: Refer to the example under BOOLEAN

BASIC Equivalent: None

UNLOCK PROCEDURE UNLOCK(Filespec);

This procedure is used to unlock a disk file which was previously locked. 'Filespec' specifies the name of the file to be unlocked. It may be either a variable of type string or a string literal (within quotes).

Example:
```
PROGRAM UNLOCK_DEMO;
VARIABLE FILENAME:STRING;

BEGIN
  FILENAME:='D:TEST.TXT';
  UNLOCK(FILENAME)
END.
```

BASIC Equivalent: XIO 36,#1,0,0,"D:TEST.TXT"

VAL FUNCTION VAL(Svar):INTEGER or REAL;

This function is used to return the value of a string variable which contains a number. 'Svar' must be a string type variable. The number must start at the beginning of the string variable. REAL values are returned to REAL variables, and INTEGER values are returned to INTEGER variables.

Example:
```
PROGRAM VAL_DEMO;
VAR I:INTEGER;
    R:REAL;
    S:STRING;

BEGIN
  S:='1234';
  I:=VAL(S);
  WRITELN('VAL(S)=',I);
  S:='12.34';
  R:=VAL(S);
  WRITELN('VAL(S)=',R)
END.
```

BASIC Equivalent: I=VAL(S$)

VAR
```
VAR Name1,Name2,...,Namen : Type;
VAR Name1,Name2,...,Namen : ARRAY[Number]
    OF Type;
```

VAR is used to allocate variables to be used by a program. Variables which are defined at the beginning of a program, before procedures and functions, are global and may be referenced by any statement in the program. Variables which are defined within procedures and functions are local variables and may only be referenced by statements within those procedures and functions. Valid 'Type' codes are FILE, CHAR, INTEGER, REAL, BOOLEAN, RECORD, and STRING. Refer to the descriptions of the individual type codes for more information about them. ARRAYs may be specified for any type other than FILE or RECORD. Refer to the description under ARRAY for more information. The variable names may be any words that begin with a letter and are not the same as Pascal reserved words. The name may be of any length, but only the first eight characters are significant and must be unique. A section listing Pascal reserved words is included within this manual.

Example: Refer to the example under STRING.

BASIC Equivalent: None for files. DIM for strings and arrays. None required for numbers.

WAIT PROCEDURE WAIT(Number);

This special built-in procedure is used to suspend program execution for a specified length of time. 'Number' is the number of sixtieths of a second for the program to wait. A value of 60 is equal to 1 second. 'Number' may be any integer expression.

Example: Refer to example under SOUND

BASIC Equivalent: None

WHILE WHILE Condition DO Statement;

WHILE is used to repeat execution of a statement until a specified condition is false. 'Condition' may be any expression which results in a true (1) or false (0) condition. The condition is evaluated before the statement is executed. If the condition is initially false, 'Statement' will not be executed.

Example:
```
PROGRAM WHILE_DEMO;
VAR INPUT:FILE;
    DATA:STRING;

BEGIN
  RESET(INPUT,'D:TEST.TXT');
  WHILE NOT EOF(INPUT) DO
    BEGIN
      READLN(INPUT,DATA);
      WRITELN(DATA)
    END;
  CLOSE(INPUT)
END.
```

BASIC Equivalent: None

WRITE
```
WRITE(File,Expr1,Expr2,...);
WRITE(File,Expr1:Fldwdth...);
WRITE(File,Expr1:Fldwdth:Numdec...);
```

The WRITE is used to move data from memory to an external device, such as the television/monitor screen, disk drive, cassette recorder, or modem. 'File' is optional and, if present, determines the device to receive the data. If 'File' is not present, then the screen is used. The variables may be of any type other than FILE. Expressions are permitted in the WRITE statement. The end-of-line character (carriage return) will not follow the data for WRITE (see WRITELN). Integer numbers with values of zero through 255 may be sent to the output device. For example, to send a form feed command to a printer (defined as file PRINTER), you can use WRITE(PRINTER,CHR(12));. Numbers by themselves will print as normal integer or real values. To write out an integer value in hexadecimal format, precede the variable name or integer value with a percent sign (%). Literal constants may be used in the WRITE statement, also. The literal must be enclosed within a pair of single quote marks. It may be any character other than a quote mark. To write a quote mark, say WRITE(CHR(39)), because 39 is the ASCII value of the quote mark.

Write formatting is supported. Refer to the example under WRITELN.

Example: Refer to example under PROCEDURE

BASIC Equivalent: PRINT (followed by a semicolon)

WRITELN

```
WRITELN(File,Expr1,Expr2,...);
WRITELN(File,Expr1:Fldwdth...);
WRITELN(File,Expr1:Fldwdth:Numdec...);
```

The WRITELN is identical to the WRITE except that an end-of-line character is sent to the output device after the variables (if present) have been written. If no expressions are present then only the end-of-line character is written. If all parameters and the parenthesis are missing, then an end-of-line character is written to the screen.

Write formatting is supported. It is handled differently, depending on the type of data to be written. To cause formatting to happen, follow the expression with a colon (:) and then an integer expression, 'Fldwdth'. If the colon is not present, then the value of the expression will be written with a field width equal to the number of character positions that the data represents.

For integer values, 'Fldwdth' specifies The minimum field width. If 'Fldwdth' is greater than the number of digits in the integer value, the value is right justified in a field containing 'Fldwdth' positions. If 'Fldwdth' is less than the number of digits in the integer value, the width of the field is increased to contain the full integer value.

For character data, 'Fldwdth' specifies the absolute field width. The character will be right justified within the field.

For string data, 'Fldwdth' specifies the maximum field width. If 'Fldwdth' is greater than the number of characters in the string, the string is right justified in a field containing 'Fldwdth' positions. If 'Fldwdth' is less than the number of characters in the string, then the string value will be truncated on the right and only 'Fldwdth' characters will be written.

For real data, 'Fldwdth' performs the same as with integer data, but 'Numdec' is permitted. If the second colon (:) and 'Numdec' are both omitted, then the real value will be printed in scientific notation. When the second colon and 'Numdec' are present, the real value is not printed in scientific notation, and 'Numdec' specifies the number of decimal positions to be printed. 'Numdec' may be any integer value from 0 through 254. If 'Numdec' is greater than the number of significant decimal positions in the value, then zeros are added on the right until 'Numdec' decimal positions are taken. If 'Numdec' is less than the number of significant decimal positions in the value, then the value written is truncated (not rounded) after 'Numdec' decimal positions.

In the example that follows, a blank is represented by a lowercase letter b.

Example:

```
PROGRAM WRITELN_DEMO;
VAR I:INTEGER;
    R:REAL;
    C:CHAR;
    S:STRING[4];
BEGIN
 I:=1234;
 R:=1.234;
```

```
C:='A';
S:='ABC'

WRITELN(I);        (* gives 1234                 *)
WRITELN(I,I);      (* gives 12341234             *)
WRITELN(I:1);      (* gives 1234                 *)
WRITELN(I:7);      (* gives bbb1234              *)

WRITELN(R);        (* gives 1.23400000E+00       *)
WRITELN(R:7);      (* gives 1.23400000E+00       *)
WRITELN(R:16);     (* gives bb1.23400000E+00     *)
WRITELN(R:7:0);    (* gives bbbbb1.              *)
WRITELN(R:7:1);    (* gives bbbb1.2              *)
WRITELN(R:7:5);    (* gives 1.23400              *)
WRITELN(R:2:5);    (* gives 1.23400              *)

WRITELN(C);        (* gives A                    *)
WRITELN(C,C);      (* gives AA                   *)
WRITELN(C:1);      (* gives A                    *)
WRITELN(C:3);      (* gives bbA                  *)

WRITELN(S);        (* gives ABC                  *)
WRITELN(S,S);      (* gives ABCABC               *)
WRITELN(S:1);      (* gives A                    *)
WRITELN(S:3);      (* gives ABC                  *)
WRITELN(S:5);      (* gives bbABC                *)

END.
```

BASIC Equivalent: PRINT (not followed by a semicolon)

XCTL PROCEDURE XCTL(Filespec);

This special built-in procedure is used to transfer control to another Pascal program. 'Filespec' may be either a string variable or a string literal (within quotes). It must completely specify the P-code to be executed next. This means that the '.PCD' extension must be present in the filename. If data is to be passed from the current program to the next program, then it must first be stored somewhere (like disk) by the current program and retrieved by the next program. If the program to be transferred to is not on the diskette currently in the drive specified, a message is given asking you to insert the correct diskette.

Example:
```
PROGRAM XCTL_DEMO;

BEGIN
  XCTL('D:NEXT.PCD')
END.
```

BASIC Equivalent: RUN "D:NEXT"

XIO PROCEDURE XIO(Number,File,Aux1,Aux2,Filespec);

XIO is used to perform special input/output operations. It may be used with any device. One use is to fill an area on the screen between plotted points and lines with a specific color. 'Number' is an integer number with a value in the range 0-255. The number specified depends on the operation requested and the device. 'File' may be either an absolute IOCB number (preceded by a #) or a variable of type FILE. 'Aux1' and 'Aux2' are auxiliary control codes and are dependant on the particular device and command number. 'Filespec' supplies the file specification to the device handler. It may be either a string variable or a string

literal (within quotes). The standard values for 'Number' are as follows:

```
3      OPEN
5      GET RECORD
7      GET CHARACTERS
9      PUT RECORD
11     PUT CHARACTERS
12     CLOSE
13     STATUS REQUEST
17     DRAW LINE
18     FILL
32     RENAME
33     DELETE
35     LOCK FILE
36     UNLOCK FILE
37     POINT
38     NOTE
254    FORMAT
```

Example:
```
PROGRAM XIO_FILL_DEMO;

BEGIN
  MAXGRAPH(5);
  GRAPHICS(5);
  COLOR(3);
  PLOT(70,45);
  DRAWTO(50,10);
  DRAWTO(30,10);
  GOTOXY(10,45);
  POKE(765,3);
  XIO(18,#6,0,0,'S:')
END.
```

BASIC Equivalent: XIO 18,#6,0,0,"S:"

System Information

The Supervisor uses zero page locations $A0 - $BF. Locations $80 - $9F are available for your use if desired. Various locations between $D4 and $FD are used by the floating point routines. Page six ($600 - $6FF) is available for your use and not used by the Pascal system.

The Supervisor is loaded into memory by DOS at the address $1D7C. If this memory location is not available, then an error message is given, along with an explanation of the probable cause of the problem. The pseudo code program to be executed is loaded in memory immediately after the end of the Supervisor. The pseudo machine stack extends from the end of the pseudo code program to the MEMTOP position, just before screen memory.

Filename Descriptions

The files named below are included in this ARC file:

AUTORUN.SYS	Supervisor object code
COMPILER.PCD	Compiler pcode
EDITOR.PCD	Editor pcode
INIT.PCD	Main Menu pcode
INIT.PAS	Main Menu Pascal source
EXPLNERR.PCD	Error code explainor (used by Compiler)
RSVDWRDS.TXT	Reserved word list (used by Compiler)
ERRORS.TXT	Text for compile errors (used by EXPLNERR.PCD)
RAMDISK1.DAT	Ramdisk setup (See "*Ramdisk Support*")
RAMDISK2.DAT	Ramdisk setup (See "*Ramdisk Support*")
RAMDISK3.DAT	Ramdisk setup (See "*Ramdisk Support*")
NOTITLE.OBJ	Used to suppress title (See "*Suppressing the Title Screen*")
SAMPLE1.PAS	Kaleidoscope sample program source
SAMPLE2.PAS	Roman numeral sample program source
PASCAL.DOC	Introduction manual

Internal Data Formats

Variables are allocated on the stack. Variables of type FILE reserve no space on the stack. The others are allocated as follows:

VAR X:BOOLEAN	2 bytes
VAR X:INTEGER	2 bytes
VAR X:ARRAY[n] OF INTEGER	2 * (n + 1) bytes
VAR X:CHAR	2 bytes
VAR X:ARRAY[n] OF CHAR	2 * (n + 1) bytes
VAR X:REAL	6 bytes
VAR X:ARRAY[n] OF REAL	6 * (n + 1) bytes
VAR X:STRING[a]	(Explained below)
VAR X:ARRAY[n] OF STRING[a]	(Explained below)

Strings and string arrays have exactly the same format internally. The first two bytes hold the actual number of elements in the string array. If it is not an array, this value is 1. The next two bytes tell the maximum length of a string entry. This ends the fixed part of string allocations. The remaining parts are repeated for as many times as there are entries in the array. Only one set is present for non-array string definitions. There is a one byte long prefix which shows the actual length of that particular string entry. It is followed immediately by the data of the string. If the maximum length of the string entries is an even number, then a one byte filler byte is added to the end of each string entry. This is required because the variables are stored on

the stack and the stack width is two bytes. Non-array strings of 80 and 81 bytes long each, would each take up 86 bytes total. A two element string array of length 10 would require 28 bytes total.

Records take up no extra displacement. They are used at compile time to specify the range of fields to be included within the record.

Suppressing the Title Screen

If you desire not to have the initial title screen displayed, the following procedure will suppress it. Make the following modifications, using DOS, to the desired diskette:

1. Unlock the file NOTITLE.OBJ.
2. Use the DOS copy function (C) to append the Supervisor (AUTORUN.SYS) to the special prefix (NOTITLE.OBJ). Enter the following when prompted for the filenames to be copied:

AUTORUN.SYS,NOTITLE.OBJ/A

The '/A' is required and instructs DOS to append the file.

3. Unlock AUTORUN.SYS.
4. Rename AUTORUN.SYS to something else (like AR.SYS).
5. Rename NOTITLE.OBJ to be AUTORUN.SYS.

Trace Format

A few lines of trace information would look like the following:

```
PC=0186 IN=20 04 00 00 SP=3DE0 SV=0000
PC=018A IN=02 88 13    SP=3DE0 SV=0020
PC=018D IN=10 0C       SP=3DE2 SV=8813
PC=018F IN=60 07 00    SP=3DE0 SV=0100
PC=0192 IN=10 00       SP=3DDE SV=0020
```

The 'PC' stands for program counter. It actually refers to the offset of the instruction to be executed. This corresponds to the offset shown on the left side of the compile listing. The 'IN' stands for instruction. The one to four bytes following it are the actual hex values of the pseudo code to be executed next. 'SP' stands for stack pointer. It is the actual address of the current location on the stack. 'SV' is stack value. The stack width is two bytes, so two bytes are shown. The actual meanings of the various pseudo instruction codes are not included with this manual but may become available in the future.

Reserved Word List

ABS	DEG	FUNCTION	ODD	READLN	STRING
ADDR	DELETE	GOTOXY	OF	REAL	THEN
AND	DIV	GRAPHICS	OPEN	RECORD	TO
ARCTAN	DO	HIMEM	OPTIONKEY	REPEAT	TRACEOFF
ARRAY	DOS	IF	OPTIONS	RESET	TRACEON
ASC	DOWNTO	INSERT	OR	REWRITE	TRUE
BEGIN	DRAWTO	INTEGER	ORD	RND	UNLOCK
BLOAD	DUMPSTK	IORESULT	PADDLE	SELECTKEY	UNTIL
BOOLEAN	DVSTAT	KEYPRESS	PEEK	SETCOLOR	VAL
CALL	ELSE	LENGTH	PLOT	SHL	VAR
CASE	END	LN	POINT	SHR	WAIT
CHAR	EOF	LOCATE	POKE	SIN	WHILE
CHR	EOLN	LOCK	POS	SOUND	WRITE
CLOSE	EXIT	LOG	PROC	SQR	WRITELN
COLOR	EXP	LPENH	PROCEDURE	SQRT	XCTL
CONCAT	EXP10	LPENV	PROGRAM	STARTKEY	XIO
CONST	FALSE	MAXGRAPH	PTRIG	STATUS	
COPY	FILE	MOD	PURGE	STICK	
COS	FOR	NOT	RAD	STR	
CVTREAL	FUNC	NOTE	READ	STRIG	

Operators

Operator	Operation
:=	assignment
arithmetic:	
+	addition
-	subtraction
*	multiplication
/ or DIV	division
MOD	modulo (remainder after division)
Relational:	
=	equality
<>	inequality
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
Logical:	
OR	
AND	
NOT	

Editor Command Summary

A Add lines to end of file in memory. Terminate add mode by entering a null line.

C Change string of characters in one or more lines.

D Delete one or more lines.

E Edit one or more lines. Make change directly on the line presented.

F Filer commands

A Append disk file to end of file currently in memory.

D List disk directory on screen.

L Load disk file into memory. Anything currently in memory will be erased.

S Save file currently in memory onto disk.

I Insert before line number you specify. Terminate insert mode by entering a null line.

L List lines from memory on the screen.

M Display Editor menu

P Print one or more lines on printer (P:).

Q Quit Editor execution and return to Main Menu screen.

S Scan one or more lines for character string you specify.

X Exit directly to the Compiler.

? Display Editor menu.

Error Messages

Compile Time Error Messages

01: Compiler table overflow (max 170)

02: Number expected

03: '=' expected

04: Identifier expected

05: Constant type identifier, number, or string constant expected

06: 'BEGIN' expected

07: Too many nesting levels

08: ':' expected

09: '.' expected

10: ';' expected

11: Undeclared identifier

12: Invalid type of identifier

13: ':=' expected

14: 'END' expected

15: ';', 'ELSE', or 'END' expected

16: 'THEN' expected

17: '#' expected

18: 'DO' expected

19: '#' or FILE type identifier expected

20: '[' expected

21: ']' expected

22: ')' expected

23: Illegal factor or identifier type

24: INCLUDE file nesting too deep

25:

26: 'OF' expected

27: Mismatched data types

28: 'TO' or 'DOWNTO' expected

29: 'UNTIL' expected

30: Range error

31: '(' expected

32: ',' expected

33: Literal too long or missing end quote (')

34: 'END' but no RECORD started

35: Incorrect number of parameters

36: INTEGER type identifier expected

37: STRING type identifier expected

38: REAL type identifier expected

39: CHAR type identifier expected

40: FILE type identifier expected

41: HEX type identifier expected

42: STRING constant expected

Execution Time Error Messages

INDEX TOO HIGH

This message occurs if an attempt is made to store a string array element into an occurance that is higher than defined for the variable. For example, if you tried to store the twentieth entry of an array that was only defined to hold ten occurances, you would get the message. This message only applies to string arrays since other array types are not checked for valid occurance numbers.

UNABLE TO OPEN DEBUG IOCB (7)

This message is issued if the list output device you specify in response to the 'WHERE? (FILESPEC)' prompt cannot be opened. The prompt is issued only for the debug features trace and stack display.

CIO ERROR xxx FOR IOCB # y

Some kind of Input-Output operation was performed which resulted in an abnormal return code from the Atari operating system. Refer to your BASIC or DOS manual for the meaning of the error number 'xxx'. 'y' is the IOCB number which the error occured on. Note that this message will not be printed if OPTIONS(0) is in effect. In this case it is your responsibility to check the return code by interrogating IORESULT after each I/O type instruction.

AT OFFSET

This message accompanies some other error message and refers to the offset within the pseudo code of the instruction that had the error. Refer to the offset shown on your compile listing to determine the Pascal instruction that experienced the error.

STOPPED BY <BREAK> KEY

This message indicates that execution of the program was stopped because the BREAK key was pressed. The offset of the instruction executing is shown in the 'AT OFFSET' message. Note that this message will not occur (and the program will not stop after BREAK is pressed) if OPTIONS(4) is in effect.

INSUFFICIENT MEMORY

This message indicates that an attempt was made to increase the value of the stack pointer to a value which would overlay screen memory or the trace buffer, if the trace was active. It may also be caused by manipulation of a record without sufficient room between the top of the stack and the top of available

memory (MEMTOP) to temporarily hold it.

INVALID OPCODE

This message should not occur. It indicates that a pseudo instruction was encountered which is invalid, If you get this message, it means that your '.PCD' file has been corrupted somehow or an XCTL was made to a file that was not a pseudo code file. To correct, re-compile the program in question. It may also occur if you attempt to run a Draper Pascal program which was compiled under a previous release of this software.

Main Menu Program

```
(* INITIAL MENU PROGRAM *)
(* AS OF 09/08/89       *)
(*$S+*)
PROGRAM INIT;
CONST CLEAR=125; CURSOR=752;
      ON=0; OFF=1;
      RAMTOP=$6A;
      LASTFILE=$1D82;
      DEFAULT_DRV=$1D94;
VAR BASENAME:STRING[11];
    PGMNAME:STRING[30];
    DATA:STRING[128];
    I,J:INTEGER;
    REPLY,DRIVENO:CHAR;
    DRIVE:STRING[3];

PROCEDURE PRESSANY;
BEGIN
    WRITELN;
    WRITE('Press any key to continue');
    READ(REPLY)
  END;

BEGIN (*MAIN*)
 DRIVENO:=PEEK(DEFAULT_DRV);
 DRIVE:=CONCAT('D',DRIVENO,':');
 GRAPHICS(0);
 POKE(CURSOR,OFF);
 WRITE(CHR(CLEAR));
 GOTOXY(2,0);
 WRITELN('             DRAPER PASCAL');
 WRITELN;
 WRITELN('              VERSION 2.1');
 WRITELN;
 WRITELN('         1 - Run Program');
 WRITELN;
 WRITELN('         2 - Disk Directory');
 WRITELN;
 WRITELN('         3 - Compile Program');
 WRITELN;
 WRITELN('         4 - Edit a Program');
 WRITELN;
 WRITELN('         5 - Exit to DOS');
 WRITELN;
 WRITELN('         6 - List a file');
 WRITELN;
 WRITELN('         7 - Trace on');
 GOTOXY(2,22);
 WRITELN('           Copyright 1989');
 WRITE('         by Norm Draper');
 GOTOXY(2,18);
 REPEAT READ(REPLY)
 UNTIL (REPLY>'0') AND (REPLY<'8');
 CASE REPLY OF
  '1': BEGIN (* Run Program *)
    REPEAT
      WRITELN('  Enter name of program to be run');
      WRITELN;
      POKE(CURSOR,ON);
      J:=ADDR(PGMNAME);
      FOR I:=0 TO 16 DO
        POKE(J+I,PEEK(LASTFILE+I));
      WRITE('        ');
      WRITELN(PGMNAME);
      WRITE('        ',CHR(28));
      READLN(BASENAME);
      J:=ADDR(BASENAME);
      FOR I:=0 TO 16 DO
```

```
          POKE(LASTFILE+I,PEEK(J+I));
        I:=POS('.',BASENAME);
        IF I<>0 THEN
          BEGIN
            J:=LENGTH(BASENAME);
            DELETE(BASENAME,I,J-I+1);
            WRITELN;
            WRITELN('Suffix not required, ignored');
            WRITELN
          END;
        IF POS(':',BASENAME)=0 THEN
          INSERT(DRIVE,BASENAME,1);
        PGMNAME:=CONCAT(BASENAME,'.PCD');
        OPTIONS(0);
        OPEN(#4,4,0,PGMNAME);
        I:=IORESULT;
        CLOSE(#4);
        IF I=170 THEN
          WRITELN('Program not found');
        OPTIONS(1)
      UNTIL I=0;
      WRITE(CHR(CLEAR));
      XCTL(PGMNAME) END;
    '2': BEGIN (* Disk Directory *)
      CLOSE(#1);
      POKE(CURSOR,ON);
      WRITE('Filespec? ');
      READLN(DATA);
      POKE(CURSOR,OFF);
      IF DATA='' THEN
        DATA:=CONCAT(DRIVE,'*.*');
      IF POS(':',DATA)=0 THEN
        INSERT(DRIVE,DATA,1);
      IF POS(':',DATA)=LENGTH(DATA) THEN
        INSERT('*.*',DATA,LENGTH(DATA)+1);
      OPEN(#1,6,0,DATA);
      READLN(#1,PGMNAME);
      WRITE(CHR(CLEAR));
      WHILE NOT EOF(#1) DO
        BEGIN WRITELN(PGMNAME);
        READLN(#1,PGMNAME) END;
      CLOSE(#1);
      PRESSANY;
      PGMNAME:=CONCAT(DRIVE,'INIT.PCD');
      XCTL(PGMNAME) END;
    '3': BEGIN (* Compile Program *)
           MAXGRAPH(0);
           WRITELN('Loading Compiler...');
           POKE(CURSOR,ON);
           PGMNAME:=CONCAT(DRIVE,'COMPILER.PCD');
           XCTL(PGMNAME)
         END;
    '4': BEGIN (* Edit a Program *)
           MAXGRAPH(0);
           WRITELN('Loading Editor...');
           PGMNAME:=CONCAT(DRIVE,'EDITOR.PCD');
           XCTL(PGMNAME)
         END;
    '5': BEGIN (* Exit to DOS *)
           POKE(CURSOR,ON);
           DOS
         END;
    '6': BEGIN (* List a file *)
      WRITELN('Enter filename of file to be listed');
      POKE(CURSOR,ON);
      WRITELN;
      READLN(PGMNAME);
      POKE(CURSOR,OFF);
      IF POS(':',PGMNAME)=0 THEN
        INSERT(DRIVE,PGMNAME,1);
```

```
      CLOSE(#1);
      OPEN(#1,4,0,PGMNAME);
      READLN(#1,DATA);
      WRITE(CHR(CLEAR));
      WHILE NOT EOF(#1) DO
        BEGIN WRITELN(DATA);
        READLN(#1,DATA) END;
      CLOSE(#1);
      PRESSANY;
      PGMNAME:=CONCAT(DRIVE,'INIT.PCD');
      XCTL(PGMNAME) END;
    '7': BEGIN (* Trace on *)
      POKE(CURSOR,ON);
      WRITELN('Enter number of entries to maintain: ');
      READ(I);
      TRACEON(I);
      PGMNAME:=CONCAT(DRIVE,'INIT.PCD');
      XCTL(PGMNAME) END
  END
END.
```

Editor Program

Editor Program Source Listings

(* EDITOR.PAS *)

```
PROGRAM EDITOR;
(* A part of Draper Pascal *)
(* By Norm Draper          *)
(* As of 09/26/86          *)
(*$S+*)
(*$I D:EDITOR1.PAS *)
(*$I D:EDITOR2.PAS *)
(*$I D:EDITOR3.PAS *)
(*$I D:EDITOR4.PAS *)
(*$I D:EDITOR5.PAS *)
(*$I D:EDITOR6.PAS *)
(*$I D:EDITOR7.PAS *)
```

(* EDITOR1.PAS *)

```
CONST CLEAR=125;BELL=$FD;ESC=27;
      UP=28;RIGHT=31;
      MAXLINES=251;MAXLENGTH=80;
      RAMTOP=$6A;
      LMARGIN=82;
      CURSOR_HORIZ=$55;
      DEFAULT_DRV=$1D94;
VAR CMD:CHAR;
    I,CHGSW,LM0,LM1,LASTLINE,LOW,HIGH,X,Y,SW:INTEGER;
    FILENAME,PGMNAME:STRING[30];
    DRIVE:STRING[3];
    DRIVENO:CHAR;
    DATA,DATA1,DATA2:STRING[MAXLENGTH];
    INPUT,OUTPUT:FILE;
    T:ARRAY[MAXLINES] OF STRING[MAXLENGTH];

PROCEDURE MENU;
BEGIN
  WRITE(CHR(CLEAR));
  WRITELN('           DRAPER SOFTWARE');
  WRITELN('               EDITOR');
  WRITELN;
  WRITELN('       A - Add line(s) at end');
  WRITELN('       C - Change line(s)');
  WRITELN('       D - Delete line(s)');
  WRITELN('       E - Edit line(s)');
  WRITELN('       F - Filer menu');
  WRITELN('       I - Insert before line');
  WRITELN('       L - List line(s)');
  WRITELN('       M - Menu');
  WRITELN('       P - Print line(s)');
  WRITELN('       Q - Quit');
  WRITELN('       S - Scan line(s)');
  WRITELN('       X - Exit to Compiler')
END;
PROCEDURE SHOWLINE(NUMBER);
BEGIN
  IF NUMBER<100 THEN WRITE(OUTPUT,' ');
  IF NUMBER<10 THEN WRITE(OUTPUT,' ');
  WRITE(OUTPUT,NUMBER,':')
END;
PROCEDURE GETDATA(NUMBER);
BEGIN
  SHOWLINE(NUMBER);
  POKE(LMARGIN,LM1);
```

```
   READLN(DATA);
   IF IORESULT=137 THEN
     WRITELN(CHR(BELL),'Line ',NUMBER,' truncated');
   POKE(LMARGIN,LM0);
   POKE(CURSOR_HORIZ,LM0)
END;

                         (* EDITOR2.PAS *)

PROCEDURE INC_LASTLINE;
BEGIN
   LASTLINE:=LASTLINE+1;
   IF LASTLINE>MAXLINES THEN
     BEGIN
       WRITELN('BUFFER FULL - STANDBY');
       LASTLINE:=MAXLINES
     END
END;
PROCEDURE GETRANGE;
BEGIN
   LOW:=1;
   HIGH:=LASTLINE;
   WRITE('Line from -> ');
   READLN(DATA);
   IF DATA<>'' THEN LOW:=VAL(DATA);
   IF LOW<1 THEN LOW:=1;
   IF LOW>LASTLINE THEN LOW:=LASTLINE;
   WRITE('Line to   -> ');
   READLN(DATA);
   IF DATA<>'' THEN HIGH:=VAL(DATA);
   IF HIGH<LOW THEN HIGH:=LOW;
   IF HIGH>LASTLINE THEN HIGH:=LASTLINE
END;
PROCEDURE EDIT;
BEGIN
   CHGSW:=1;
   GETRANGE;
   FOR I:=LOW TO HIGH DO
     BEGIN
       SHOWLINE(I);
       POKE(LMARGIN,LM1);
       WRITELN(T(I));
       FOR Y:=0 TO LENGTH(T(I)) / (40-LM1) DO
         WRITE(CHR(UP));
       READLN(T(I));
       POKE(LMARGIN,LM0);
       POKE(CURSOR_HORIZ,LM0)
     END
END;
PROCEDURE GETFN;
CONST LASTFILE=$1D82;
BEGIN
   WRITE('Enter filename  -> ');
   Y:=ADDR(DATA);
   FOR X:=0 TO 16 DO
     POKE(Y+X,PEEK(LASTFILE+X));
   WRITELN(DATA);
   WRITE('                    ',CHR(UP));
   READLN(FILENAME);
   Y:=ADDR(FILENAME);
   FOR X:=0 TO 16 DO
     POKE(LASTFILE+X,PEEK(Y+X));
   IF POS(':',FILENAME)=0 THEN
     INSERT(DRIVE,FILENAME,1);
   I:=LENGTH(FILENAME);
   IF POS('.',FILENAME)=0 THEN
     INSERT('.PAS',FILENAME,I+1)
END;
```

(* EDITOR3.PAS *)

```
PROCEDURE GETONE;
BEGIN
  WRITE('Line -> ');
  READ(LOW);
  IF LOW<1 THEN LOW:=1
END;
PROCEDURE SAVE;
BEGIN
  GETFN;
  Y:=0;
  REWRITE(OUTPUT,FILENAME);
  X:=IORESULT;
  IF X<>0 THEN
    Y:=X
  ELSE
    FOR I:=1 TO LASTLINE DO
      BEGIN
        WRITELN(OUTPUT,T(I));
        X:=IORESULT;
        IF X<>0 THEN
          Y:=X
      END;
  REWRITE(OUTPUT,'E:');
  OPTIONS(4); (* DISABLE BREAK KEY *)
  IF Y<>0 THEN
    WRITELN(CHR(BELL),'***Error ',Y,' while saving to disk');
  CHGSW:=0
END;
PROCEDURE CHECKUPD;
BEGIN
  IF CHGSW=1 THEN
    BEGIN
      WRITELN('File changed but not saved');
      WRITELN('Enter "I" to IGNORE and continue');
      WRITELN('   or "S" to SAVE   and continue');
      REPEAT
        READ(CMD)
      UNTIL (CMD='I') OR (CMD='S');
      IF CMD='S' THEN SAVE
    END
END;
PROCEDURE KEYBOARD;
BEGIN
  IF SW=1 THEN
    REPEAT
      SW:=SW
    UNTIL KEYPRESS;
  IF KEYPRESS THEN
    BEGIN
      READ(CMD);
      IF CMD=' ' THEN
        SW:=1
      ELSE
        SW:=0
    END
END;
```

(* EDITOR4.PAS *)

```
PROCEDURE APPEND;
BEGIN
  REPEAT
    INC_LASTLINE;
```

```
    GETDATA(LASTLINE);
    T(LASTLINE):=DATA
  UNTIL DATA='';
  CHGSW:=1;
  LASTLINE:=LASTLINE-1
END;
PROCEDURE COMPILE;
BEGIN
  CHECKUPD;
  CLOSE(OUTPUT);
  MAXGRAPH(0);
  WRITELN;
  WRITELN('Loading Compiler ...');
  OPTIONS(5); (* ENABLE BREAK KEY *)
  PGMNAME:=CONCAT(DRIVE,'COMPILER.PCD');
  XCTL(PGMNAME)
END;
PROCEDURE DLTE;
BEGIN
  GETRANGE;
  FOR I:=0 TO LASTLINE-HIGH-1 DO
    BEGIN
      X:=LOW+I;
      Y:=HIGH+1+I;
      T(X):=T(Y)
    END;
  CHGSW:=1;
  LASTLINE:=LASTLINE-(HIGH-LOW)-1
END;
PROCEDURE DIRECTORY;
BEGIN
  WRITE('Filespec? ');
  READLN(DATA);
  IF DATA='' THEN
    DATA:=CONCAT(DRIVE,'*.*');
  IF POS(':',DATA)=0 THEN
    INSERT(DRIVE,DATA,1);
  IF POS('.',DATA)=0 THEN
    INSERT('*.*',DATA,LENGTH(DATA)+1);
  OPEN(#5,6,0,DATA);
  READLN(#5,DATA);
  REPEAT
    WRITELN(DATA);
    READLN(#5,DATA)
  UNTIL EOF(#5);
  CLOSE(#5)
END;

                    (* EDITOR5.PAS *)

PROCEDURE INSRT;
BEGIN
  CHGSW:=1;
  GETONE;
  GETDATA(LOW);
  WHILE DATA<>'' DO
    BEGIN
      FOR I:=LASTLINE DOWNTO LOW DO
        BEGIN
          X:=I+1;
          T(X):=T(I)
        END;
      INC_LASTLINE;
      T(LOW):=DATA;
      LOW:=LOW+1;
      GETDATA(LOW)
    END
END;
```

```
PROCEDURE LIST;
BEGIN
   GETRANGE;
   FOR I:=LOW TO HIGH DO
      BEGIN
         SHOWLINE(I);
         POKE(LMARGIN,LM1);
         WRITELN(T(I));
         POKE(LMARGIN,LM0);
         POKE(CURSOR_HORIZ,LM0);
         KEYBOARD;
         IF CMD=ESC THEN
            I:=HIGH+1
      END
END;
PROCEDURE PRINT;
BEGIN
   GETRANGE;
   OPTIONS(0);
   REWRITE(OUTPUT,'P:');
   IF IORESULT<>0 THEN
      BEGIN
         WRITELN('PRINTER NOT READY        ');
         WRITELN('PRESS START WHEN READY');
         REPEAT
            OPTIONS(0)
         UNTIL STARTKEY;
         REWRITE(OUTPUT,'P:')
      END;
   FOR I:=LOW TO HIGH DO
      BEGIN
         SHOWLINE(I);
         WRITELN(OUTPUT,T(I))
      END;
   OPTIONS(1);
   REWRITE(OUTPUT,'E:');
   OPTIONS(4) (* DISABLE BREAK KEY *)
END;

                         (* EDITOR6.PAS *)

PROCEDURE APNDFILE;
VAR IOR:INTEGER;
BEGIN
   OPTIONS(0);
   REPEAT
      GETFN;
      RESET(INPUT,FILENAME);
      IOR:=IORESULT;
      IF IORESULT<>0 THEN
         WRITELN('File not found');
   UNTIL IOR=0;
   WHILE NOT EOF(INPUT) DO
      BEGIN
         INC_LASTLINE;
         READLN(INPUT,T(LASTLINE));
         IF IORESULT=137 THEN
            WRITELN(CHR(BELL),'Line ',LASTLINE,' truncated')
      END;
   LASTLINE:=LASTLINE-1;
   CLOSE(INPUT);
   WRITELN(LASTLINE,' lines now in memory')
END;
PROCEDURE CHANGE;
VAR PRTSW:INTEGER;
BEGIN
   GETRANGE;
   WRITE('Change from ->');
```

```
  READLN(DATA1);
  WRITE('Change to    ->');
  READLN(DATA2);
  Y:=LENGTH(DATA1);
  FOR I:=LOW TO HIGH DO
    BEGIN
      DATA:=T(I);
      PRTSW:=0;
      X:=POS(DATA1,DATA);
      IF POS(DATA1,DATA)<>0 THEN
        BEGIN
          CHGSW:=1;
          PRTSW:=1;
          DELETE(DATA,X,Y);
          INSERT(DATA2,DATA,X);
          T(I):=DATA
        END;
      IF PRTSW=1 THEN
        BEGIN
          SHOWLINE(I);
          WRITELN(DATA);
          KEYBOARD
        END
    END
END;
PROCEDURE SCAN;
BEGIN
  GETRANGE;
  WRITE('Scan for   ->');
  READLN(DATA1);
  FOR I:=LOW TO HIGH DO
    BEGIN
      DATA:=T(I);
      IF POS(DATA1,DATA)<>0 THEN
        BEGIN
          SHOWLINE(I);
          WRITELN(DATA);
          KEYBOARD;
          IF CMD=ESC THEN
            I:=HIGH+1
        END
    END
END;
```

(* EDITOR7.PAS *)

```
PROCEDURE FILER;
BEGIN
  WRITELN('       A - Append file');
  WRITELN('       D - Directory list');
  WRITELN('       L - Load file');
  WRITELN('       S - Save file');
  REPEAT
    READ(CMD)
  UNTIL (CMD='A')
     OR (CMD='D')
     OR (CMD='L')
     OR (CMD='S');
  CASE CMD OF
    'A':BEGIN
          IF LASTLINE>0 THEN CHGSW:=1;
          APNDFILE
        END;
    'D':DIRECTORY;
    'L':BEGIN
          CHGSW:=0;
          LASTLINE:=0;
          APNDFILE
```

```
        END;
      'S':SAVE
    END
  END;
BEGIN (* MAIN *)
  DRIVENO:=PEEK(DEFAULT_DRV);
  DRIVE:=CONCAT('D',DRIVENO,':');
  CHGSW:=0;
  REWRITE(OUTPUT,'E:');
  MENU;
  OPTIONS(4); (* DISABLE BREAK KEY *)
  LM0:=PEEK(LMARGIN);
  LM1:=LM0+4;
  REPEAT
    WRITE('A,C,D,E,F,I,L,M,P,Q,S,X,?->');
    READ(CMD);
    WRITELN(CMD);
    CASE CMD OF
      'A':APPEND;
      'C':CHANGE;
      'D':DLTE;
      'E':EDIT;
      'F':FILER;
      'I':INSRT;
      'L':LIST;
      'M','?':MENU;
      'P':PRINT;
      'Q':;
      'S':SCAN;
      'X':COMPILE
    ELSE
      WRITELN(CHR(BELL),'Invalid command')
    END
  UNTIL CMD='Q';
  CHECKUPD;
  CLOSE(OUTPUT);
  OPTIONS(5); (* ENABLE BREAK KEY *)
  PGMNAME:=CONCAT(DRIVE,'INIT.PCD');
  XCTL(PGMNAME)
END.
```

Ramdisk Programs

(* RAMDISK.PAS *)

```
PROGRAM RAMDISK;
(* As of 09/26/86 *)
(* This program checks to see if the Ramdisk has already been
   setup yet for Draper Pascal. If it has not been, then the
   required files are copied from drive one to the Ramdisk. Then
   the default drive number is changed to the Ramdisk drive
number
   and control is transferred to the Ramdisk copy of INIT.PCD.
   The Ramdisk drive number is determined by the second entry in
   the file D1:COPYLIST.TXT. *)
VAR CH:CHAR;
    RC,SUBADDR:INTEGER;
    DRIVEIN,DRIVEOUT:STRING[3];
    DATA,INPUT,OUTPUT:STRING[20];
BEGIN
 OPEN(#3,4,0,'D1:COPYLIST.TXT');
 READLN(#3,DRIVEIN);   (* Input drive *)
 READLN(#3,DRIVEOUT);  (* Output (Ramdisk) drive *)
 OUTPUT:=CONCAT(DRIVEOUT,'COMPILER.PCD');
 OPTIONS(0); (* Trap errors *)
 OPEN(#1,4,0,OUTPUT);
 RC:=IORESULT;
 OPTIONS(1); (* Stop trapping errors *)
 CLOSE(#1);
 IF RC <> 0 THEN
  BEGIN
   BLOAD('D1:COPYFILE.OBJ');
   SUBADDR:=PEEK($2E1);
   SUBADDR:=SUBADDR SHL 8;
   SUBADDR:=SUBADDR+PEEK($2E0);
   WRITELN('Copying:');
   READLN(#3,DATA);
   WHILE NOT EOF(#3) DO
    BEGIN
     INPUT:=CONCAT(DRIVEIN,DATA);
     OUTPUT:=CONCAT(DRIVEOUT,DATA);
     WRITELN(INPUT,' -> ',OUTPUT);
     CLOSE(#1,#2);
     OPEN(#1,4,0,INPUT);
     OPEN(#2,8,0,OUTPUT);
     CALL(SUBADDR);
     CLOSE(#1,#2);
     READLN(#3,DATA)
    END
  END;
 CLOSE(#3);
 (* Change the default drive to the Ramdisk drive *)
 SUBADDR:=ADDR(DRIVEOUT)+2;
 POKE($1D94,PEEK(SUBADDR));
 OUTPUT:=CONCAT(DRIVEOUT,'INIT.PCD');
 XCTL(OUTPUT)
END.
```

(* COPYFILE.M65 *)

```
10        .TITLE "FILE COPY UTILITY PROGRAM"
20  ; BY NORM DRAPER
30  ;
40  ; AS OF 09/04/86
50  ;
60        .ORG $5000  ;09/04/86
70  ;    SPACE  2
80  ; Operating System Equates  ------------------------
90  ;    SPACE  1
0100 MEMLO  .EQU $02E7 ;start of user memory
0110 ICCOM  .EQU $0342 ;CIO command
```

```
0120 ICBAL   .EQU $0344
0130 ICBAH   .EQU $0345
0140 ICBLL   .EQU $0348
0150 ICBLH   .EQU $0349
0160 ICAX1   .EQU $034A
0170 ICAX2   .EQU $034B
0180 DOSINI  .EQU $0A
0190 CIOV    .EQU $E456 ;CIO vector
0200 EOL =      $9B
0210 EOF =      $88
0220 BELL =     $FD
0230 IOCB1 =    $10
0240 IOCB2 =    $20
0250 OPEN =     $03
0260 GETCHR =   $07
0270 PUTCHR =   $0B
0280 CLOSE =    $0C
0290 OREAD =    $04
0300 OWRIT =    $08
0310 ;
0320 START = *
0330      LDX #IOCB1
0340      LDA BUFFADR
0350      STA ICBAL,X
0360      STA ICBAL+16,X
0370      LDA BUFFADR+1
0380      STA ICBAH,X
0390      STA ICBAH+16,X
0400 LOOP =  *
0410      LDX #IOCB1   ;RESET IOCB PTR
0420      LDA BUFFLEN
0430      STA ICBLL,X
0440      LDA BUFFLEN+1
0450      STA ICBLH,X
0460      LDA #GETCHR
0470      STA ICCOM,X
0480      JSR CIOV
0490      STY SAVEY    ;SAVE CIO RC
0500      LDX #IOCB1   ;RESET IOCB PTR
0510      LDA ICBLL,X
0520      STA ICBLL+16,X
0530      LDA ICBLH,X
0540      STA ICBLH+16,X
0550      LDX #IOCB2   ;RESET IOCB PTR
0560      LDA #PUTCHR
0570      STA ICCOM,X
0580      JSR CIOV
0590      CPY #0
0600      BEQ OKAY
0610      RTS
0620 OKAY =  *
0630      LDY SAVEY
0640      CPY #EOF
0650      BEQ DONE
0660      CPY #0
0670      BEQ LOOP
0680 DONE =  *
0690      RTS
0700      ;
0710 SAVEY  .BYTE 0
0720 BUFFADR .WORD ENDPGM
0730 BUFFLEN .WORD $BC00-ENDPGM
0740 ENDPGM = *
0750      .ORG $02E0
0760      .WORD START
0770      .END  START
```

Sample Programs

(* SAMPLE1.PAS *)

```
PROGRAM KALEIDOSCOPE;
VAR I,J,K,W,X:INTEGER;
BEGIN
 MAXGRAPH(3);
 GRAPHICS(19);
 X:=0;
 REPEAT
 FOR W:=3 TO 50 DO
  BEGIN
    FOR I:=1 TO 10 DO
    BEGIN
     FOR J:=0 TO 10 DO
      BEGIN
       K:=I+J;
       COLOR(J*3/(I+3)+I*W/12);
       PLOT(I+8,K);
       PLOT(K+8,I);
       PLOT(32-I,24-K);
       PLOT(32-K,24-I);
       PLOT(K+8,24-I);
       PLOT(32-I,K);
       PLOT(I+8,24-K);
       PLOT(32-K,I)
      END
    END
  END
 UNTIL X=99 (* UNENDING LOOP *)
END.
```

(* SAMPLE2.PAS *)

```
PROGRAM ROMAN;
(*   ROMAN NUMERAL SAMPLE PROGRAM   *)
(*     ADAPTED FROM PASCAL USER MANUAL AND REPORT BY JENSEN AND
RTH     *)
  VAR X,Y:INTEGER;
   BEGIN Y:=1;
   REPEAT X:=Y; WRITE (X,' ');
    WHILE X>=1000 DO
     BEGIN
      WRITE ('M'); X:=X-1000
     END;
    IF X>=500 THEN
     BEGIN
      WRITE ('D'); X:=X-500
     END;
    WHILE X>=100 DO
     BEGIN
      WRITE ('C'); X:=X-100
     END;
    IF X>=50 THEN
     BEGIN
      WRITE ('L'); X:=X-50
     END;
    WHILE X>=10 DO
     BEGIN
      WRITE ('X'); X:=X-10
     END;
    IF X>=5 THEN
     BEGIN
      WRITE ('V'); X:=X-5
     END;
    WHILE X>=1 DO
```

```
     BEGIN
      WRITE ('I'); X:=X-1
     END;
  WRITELN;
  Y:=Y*2
 UNTIL Y>5000
 END.
```

Printer Usage

Printer usage with Draper Pascal

To print a Pascal source program, you can load the program into memory using the Editor, as normal. Then use the 'P' command to print on the printer. The source statements will be preceded by a line number and a colon.

There are two ways to print data from your program onto the printer. The first is similar to the way it would be done in BASIC. An example is:

```
PROGRAM PRINT_1;

VAR I,J: INTEGER;

BEGIN

  OPEN(#2,8,0,'P:');
  FOR I:=1 TO 10 DO

    BEGIN
      J:=I*10;
      WRITELN(#2,I:10,J)
    END;

  CLOSE(#2)

END.
```

The above example prints a multiplication table on the printer. The second way to print is by using a FILE type variable assigned to a printer. An example providing the same results as above is:

```
PROGRAM PRINT_2;

VAR I,J: INTEGER;
    PRINTER:FILE;

BEGIN

  REWRITE(PRINTER,'P:');
  FOR I:=1 TO 10 DO

    BEGIN
      J:=I*10;
      WRITELN(PRINTER,I:10,J)
    END;

  CLOSE(PRINTER)

END.
```

Software License

Non-Exclusive, Royalty-free License to distribute the Draper Pascal Supervisor

I. Purpose

This royalty-free, non-exclusive license is provided to allow widespread use of software developed using Draper Pascal. It applies only to the original purchaser of Draper Pascal ("Licensee").

II. The License

Subject to the conditions stated herein, Draper Software will grant to the Licensee a royalty-free, non-exclusive license to distribute the run-time system ("Supervisor"). Licensee is only authorized to distribute the Supervisor in object code form and only in conjunction with software developed by Licensee which requires the Supervisor for proper operation. Licensee shall not use or purport to authorize any person to use any of the copyrights, trademarks, service marks, or trade names of Draper Software without prior written consent from Draper Software.

The Supervisor consists of the file named AUTORUN.SYS on the supplied diskette. It may be distributed under another name if Licensee so desires.

The supplied Disk Operating System (DOS) is excluded and may not be distributed by Licensee.

III. The License Term

This license will run for a term of five (5) years from date of license acceptance. Extensions beyond that term may be secured by written permission from Draper Software.

IV. Acceptance

The term of this license will begin two weeks after Licensee has signed and returned a copy of this license to Draper Software, providing that no reject notice was sent to you by Draper Software within the two week period.

V. Additonal Terms and Conditions

A. Licensee understands and agrees that:

1. The Supervisor is distributed on an "as is" basis without warranty of any kind from Draper Software.

2. The entire risk as to the performance and quality of the Supervisor is with the Licensee.

3. If the Supervisor, as incorporated into Licensee's products proves defective following it's purchase, Licensee and not Draper Software, Draper Software's distributors, or retailers, assumes all costs associated with or resulting from use of Licensee's products including all necessary repair or servicing.

4. Draper Software shall have no liability to Licensee or to customers of Licensee for loss or damage, including consequential and/or incidental damage, caused or alleged to be caused, directly or indirectly, by the Supervisor. This includes, but is not limited to, any interruption in service or loss of business or anticipatory profits resulting from the use or operation of the Supervisor.

B. Licensee shall indemnify and hold Draper Software harmless from any claim, loss, or liability allegedly arising out of or relating to the operation of the Supervisor as used by Licensee or customers of Licensee pursuant to this license agreement.

C. Licensee shall not suggest, imply or indicate in any manner that any of Licensee's software products which incorporate or use the licensed Supervisor are approved or endorsed by Draper Software.

D. Licensee acknowledges that a failure to conform to the provisions of Section V, Subsection C (above) will cause Draper Software irreparable harm and Draper Software's remedies at law will be inadequate. Licensee acknowledges and agrees that Draper Software shall have the right, in addition to other remedies, to obtain an immediate injunction enjoining any breach of Licensee's obligations set forth in Section V.C above.

E. No waiver or modification of any provisions of this license shall be effective unless in writing and signed by the party against whom such waiver or modification is sought to be enforced. No failure or delay by either party in exercising any right, power or remedy under this license shall operate as a waiver of any such right, power or remedy.

F. This license shall bind and work to the benefit of the successors and assigns of the parties hereto. Licensee may not assign rights or delegate obligations which arise under this license to any third party without the express written consent of Draper Software. Any such assignment or delegation, without written consent of Draper Software, shall be void.

G. The validity, construction and performance of this license shall be governed by the substantive law of the State of Texas and of the United States of America excluding that body of law related to choice of law. Any action or

proceeding brought to enforce the terms of this license shall be brought in the County of Dallas, State of Texas, if under state law.

H. In the event of any legal proceeding between the parties arising from this license, the prevailing party shall be entitled to recover, in addition to any other relief awarded or granted, its reasonable costs and expenses, including attorneys' fees, incurred in the proceeding.

Your Name ____________________

Company Name (if any) ____________________

Address ____________________

City, State, Zip ____________________

Telephone Number ____________________

Signature and Date ____________________